uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería en Tecnologías Industriales

Curso académico 2017-2018

Trabajo Fin de Grado

Sistema de detección de peatones para vehículos autónomos

Francisco Javier Galarza Sánchez

Tutor

Arturo de la Escalera Hueso

Madrid 2018

RESUMEN

El objetivo de este proyecto es desarrollar software orientado a vehículos autónomos que sea capaz de tener un control numérico de los peatones en el entorno del propio vehículo. Para alcanzar esta meta, el sistema estará basado en la programación de un código de visión artificial en el lenguaje C++ que será capaz de analizar imágenes de forma automática en busca de la información necesaria para lograrlo. Esta información será principalmente la posición y la velocidad de los peatones que estén contenidos en las imágenes tomadas por propias cámaras del vehículo autónomo.

En esta memoria se presentarán todos los módulos que han sido necesarios programar para el correcto funcionamiento del producto final. Se tratará de una forma más detallada las funciones específicas de la visión artificial, pero también se abordarán temas de la estructura del código y sus motivos de una forma más general.

Por último, la parte final del documento abordara de forma detallada los resultados que se han podido sacar tras emplearlo para analizar la secuencia de imágenes que ha servido de entono de pruebas. El análisis de estos resultados se realizará tanto de forma individual a cada módulo, como al conjunto final. De esta forma se podrá juzgar las características finales de cada uno de los componentes, así como la respuesta global de todo el conjunto.

Abstract

The objective of this project is to develop an autonomous vehicle software that is capable of handling numeric control over the pedestrians in the surroundings of the autonomous car itself. In order to reach this goal, the system will be coded in C++ and based in artificial vision libraries. The system must be capable of handling information automatically so that it can get the data needed to work. This data will be primarily the location and velocity of the surrounding pedestrians contained in the images taken by the autonomous vehicle cameras.

During the development of this report, all the modules that have been necessary in order to archive the proper functioning of the final code will be broken down. Specific parts of the code referred to artificial vision will be addressed in more detail, while also more general theme related topics like the structure of the code and the decision making involved will be discussed.

Finally, the last part of the document will approach in detail the results obtained by running the code in the sequence of images that had worked as a test bench. The analysis will be conduct individually for every module, as well as the program as a whole. In that way, we could judge the final specs of each component in addition to the complete code created.

Índice de contenidos

CAPÍTULO 5: Análisis de resultados

CAPÍTULO 6: Presupuesto

CAPÍTULO 7: Conclusiones

CAPÍTULO 8: Bibliografía

Índice de figuras

Índice de tablas

CAPÍTULO 1: Introducción

1.1 Motivación del trabajo

La invención del automóvil y su posterior generalización en todo el mundo ha sido una gran revolución en la forma en la que vivimos nuestras vidas. Esta invención nos ha proporcionado la capacidad de recorrer grandes distancias en tiempos que antes parecían imposibles. Desplazamientos que parecían estar solo al alcance de grandes maquinas operadas por equipos de personas, como el ferrocarril, que habían supuesto una gran revolución y que ahora se encuentran al alcance de todo el mundo.

Pero con todo gran avance llegan nuevas dificultades que hay que superar. Entre ellas nos encontramos los accidentes. El año 2015 según el observatorio europeo de seguridad vial, The *European Road Safety Observatory*, 5516 peatones fallecieron en total solo en países de la Unión. Este número a pesar de ser elevado, solo representa los peatones fallecidos durante el año, no se incluyen accidentes entre vehículos, solo los que involucraron los viandantes. Además, este organismo estima que por cada accidente mortal se producen 4 lesiones permanentes de columna o daño cerebral, 8 heridos graves y 50 heridos leves adicionales. [1]

Con estos datos podemos ver que, aunque constantemente se están tomando medidas para intentar frenarlos, es necesario seguir avanzando en materia de prevención de acidentes.

Casi la totalidad de estos accidentes se deben a imprudencias por parte de alguno, o todos los involucrados. Es por esto, que continuamente se están implantando nuevas medidas para reducir estos incidentes, tanto a nivel legislativo como de infraestructuras mejorando la seguridad en las vías.

Sin embrago, aunque los accidentes, y con ellos la mortalidad se reducen año tras año estamos aún por encima del objetivo fijado por la Unión Europea. Y es por esto que se deben desarrollar nuevas tecnologías que permitan seguir avanzando hacia el objetivo de cero accidentes. Tecnologías que hagan uso del gran potencial de cálculo de los ordenadores del que disponemos para evitar accidentes.

1.2 Objetivos del proyecto

El objetivo de este proyecto es diseñar y desarrollar una aplicación para su uso en vehículos autónomos. Esta aplicación debe ser capaz de dotar de mayor seguridad durante la conducción al colectivo más vulnerable que circula por las vías, los peatones.

Esta aplicación debe ser capaz de identificar a los peatones en los entornos viarios y diferenciarlos del resto de elementos presentes en el entorno. Cuando esto se logre se podrá extraer la información que se necesita para poder caracterizar la posición y movimientos de los peatones. Partiendo de estos datos se podrán crear futuras aplicaciones que hagan uso de estos datos y estén enfocadas a determinar trayectorias libres de peatones, evitando así posibles colisiones.

El desarrollo de la aplicación se hará haciendo uso de un único tipo de sensor, las cámaras de imágenes convencionales. Esto tendrá como finalidad principal hacer del vehículo autónomo un sistema muy polivalente gracias a la gran capacidad de estos sistemas, ya que pueden ser utilizados para la medición de numerosos parámetros con un único tipo de sensor. Esta elección además procurará que el sistema sea fácilmente implementable debido a la gran accesibilidad de este tipo de sensores que son relativamente económicos y muy estandarizados en la industria.

Como ultima ventaja de usar únicamente las cámaras se pretende crear un sistema que sea lo más sencillo posible para así convertir el sistema en un estándar en la seguridad vial.

1.3 Marco regulador

Los vehículos autónomos son un fenómeno que está experimentando un gran crecimiento en los últimos años, pero que no existía fuera de los entornos de pruebas hasta hace muy poco. Esto provoca que se encuentren en un vacío en lo que a legislación se refiere, provocando que muchos de los códigos de circulación que los deberían regir no los tengan si quiera contemplados.

En estos momentos Estados Unidos junto a Japón son los países que ya han empezado a desarrollar regulaciones en los últimos años. Sin embargo, el caso de Estados Unidos es algo poco coordinado, ya que no tienen unas directrices comunes, cada estado puede regularlo de forma independiente. Las únicas normas que son comunes a todo el país son que no se puede prohibir su uso, y alguna otra consideración para facilitar el desarrollo de estos en el futuro próximo. Esto hace que de momento no sean el ejemplo idóneo a seguir por el resto del mundo. [2]

Como era de esperar con las tecnologías emergentes, los gobiernos no suelen estar a la par con las velocidades a las que estas aparecen. Por suerte aún hay tiempo hasta que se alcance un nivel de desarrollo elevado y se estandaricen su uso. Por eso la Unión Europea en mayo de este año 2018, publicó el comunicado sobre movilidad autónoma en las carreteras. En este documento se establecen los objetivos de conseguir unas infraestructuras adecuadas para el uso de estos vehículos. Además, se afirma la intención de alcanzar una normativa común a todos los estados miembros, para evitar los problemas de cohesión mencionados de Estados Unidos. Sin embargo, por el momento no se tiene una regulación firme y solo algunos países como Alemania, Francia o reino unido los que cuentan con algunos proyectos que están comenzado como ensayos a mayor escala con modificaciones en las infraestructuras de transporte. [3]

En nuestro país, a día de hoy no hay ninguna normativa específica para su uso. Actualmente el mayor avance con el que se cuenta es que ya es posible pedir permisos a la DGT para realizar pruebas en vías abiertas al público, y se concederán permisos acordes al nivel de automatización demostrada del vehículo que se quiera probar. [4]

Se puede ver que aún queda un largo camino por recorrer, no solo en el terreno del desarrollo y la investigación, sino también en la normativa. Esta tendrá que salir adelante lo más rápidamente posible para poder servir de guía regulando su actividad y asegurando así su estandarización.

1.4 Planificación del proyecto

Para poder desarrollar un proyecto de estas características, es necesario tener un plan sobre el que estructurar el trabajo. Esto resultara útil como guía para poder guiar el esfuerzo a las distintas tareas que tiene asociado el trabajo.

Dado que el tema de este trabajo no es algo que se trabaje durante el grado de tecnologías industriales, la visión artificial, ha sido necesaria una gran parte de investigación. Esta comienza con un curso online introductorio de la plataforma Edx. También se tendrá que buscar soluciones que nos proporcionen la base para implementar el programa que se busca crear.

Además de esto, la plataforma con la que se va a trabajar es Linux, un sistema operativo distinto a Windows. Esto también requerirá de un trabajo previo para poder adaptarse a los cambios entre ambos sistemas.

De esta forma, tendremos dividido en cinco tareas principales:

- **Introducción a la visión artificial.** Haciendo uso del mencionado curso, se pretende adquirir los conocimientos básicos para poder profundizar en ellos cuando así lo requiera el proyecto.
- **Planificación previa**. Una vez se conozcan los fundamentos de la visión artificial se puede comenzar a plantear la hoja de ruta que se va a seguir y se puede crear un diseño esquemático que represente como debería ser el producto final.
- **Instalación de programas previos.** En esta etapa se comenzará a instalar todos los programas y librerías que son necesarios para el desarrollo de la aplicación. Además, al mismo tiempo se requerirá que se aprenda a controlar el sistema operativo Linux que soportará todas estas instalaciones.
- **Desarrollo del código.** Se procederá al realizar los documentos y ejecutables necesarios para poder transformar la información de la que se parte para obtener los resultados. Esta fase contara con un factor de continuo aprendizaje, ya que cada problema al que se quiera dar solución implicara una búsqueda de información para poder encontrar una solución y aplicarla mediante el lenguaje de programación.
- **Redacción de la memoria.** Este proceso de documentación se llevará a cabo durante y tras el desarrollo de la aplicación. Se plasmarán el proceso de diseño, estructuras adoptadas, y posteriormente un análisis de los resultados obtenidos.

Debido a la incertidumbre que existe al realizar un proyecto con el que no se está muy relacionado es que la duración de las tareas tiene bastante incertidumbre. Por eso, en la planificación original se realizó usando periodos de meses para estas tareas. Estas se pueden ver representadas en la tabla a continuación.

<table>
<tr><th colspan="9">Periodo diciembre 2017 - agosto 2018</th></tr>
<tr><th colspan="2">Diciembre</th><th>Enero</th><th>Febrero</th><th>Marzo</th><th>Abril</th><th>Mayo</th><th>Junio</th><th>Julio</th><th>Agosto</th></tr>
<tr><td colspan="3">Realización curso Edx</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="3">Planificación previa</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2"></td><td></td><td>Instalación de programas previos</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2"></td><td></td><td></td><td colspan="5">Desarrollo del código</td><td></td></tr>
<tr><td colspan="2"></td><td></td><td></td><td colspan="6">Redacción de la memoria</td></tr>
</table>

Tabla 1 Planificación inicial de las tareas del proyecto

Como se puede ver, la planificación abarca desde diciembre del año 2017 hasta agosto del 2018. Este inicio se corresponde con la fecha en la que se apalabro el proyecto, y el final era la fecha estimada para su finalización. Se considero directamente la fecha de presentación de septiembre dado que era una aproximación más prudente debido a la complejidad del tema que se quería tratar.

Este proyecto cuenta con dos tareas con fechas críticas. La primera fecha es el final del proceso de instalación de programas previos. Esta que limita el inicio del desarrollo del código, y por tanto el resto del proyecto. Por este motivo, se le dedico especial atención para asegurar que se pudiera comenzar a programar el código principal del programa.

La otra fase con una fecha critica es la redacción de la memoria. No su fecha de inicio, pero si su fecha de fin, ya que atrasarla podría traer como consecuencias no poder presentar el proyecto a tiempo.

La fecha de fin del curso de introducción tenía algo más de margen, ya que se disponía hasta el inicio de la programación para terminarlo. Otra fecha también flexible es el inicio de la redacción de la memoria, ya que se podía empezar más tarde si se le dedicaba una mayor carga de trabajo, la única limitación de esta tarea era la fecha de final.

Los periodos de tiempo que finalmente fueron necesarios para el proyecto, se pueden encuentra en el *capítulo 6, el presupuesto*, donde se analiza el coste del proyecto, por lo que es necesario un análisis del tiempo que finalmente se ha empleado.

CAPÍTULO 2: Estado del arte

2.1 Introducción a la conducción autónoma

La conducción autónoma es un término que, aunque no es nuevo, si está siendo desarrollado en los últimos años de una forma más acelerada. Consiste en delegar las funciones asignadas a la conducción, que normalmente realizaría un conductor humano al propio vehículo. Éste, debe ser capaz de llegar al destino evitando los distintos obstáculos que pueda encontrar durante su trayecto, incluyendo a otros usuarios de las vías como lo son los peatones, y además de ser capaz de interactuar con el resto del tráfico rodado.

Para lograr estos objetivos, hay dos grandes partes involucradas que deben existir y cooperar.

El primero de ellos es la mecánica, el propio coche con todos sus actuadores que le permitan realizar las acciones necesarias que se le requieran. Esta parte no debería ser extremadamente difícil de implementar, ya que en la actualidad los coches ya cuentan con ordenadores de abordo que asisten en distintas tareas. Ejemplos que a día de hoy son la norma podrían ser dirección asistida que nos facilita el control de la dirección del vehículo, o sistemas de frenado inteligente como el ABS que mejora la eficiencia del mismo en situaciones límite.

Y aunque esta cuestión está lejos de ser trivial, al no ser el punto central del trabajo lo dejaremos a un lado para centrarnos más en la otra parte del sistema autónomo, el sistema de control y toma de decisiones.

Este es el sistema, que mediante procesadores de datos tomará las decisiones del vehículo como conjunto y de cada uno de sus actuadores, y que estará encargado de comunicarlas para que la parte mecánica pueda llevarlas a cabo.

Para esto, además de tener un sistema que tome las decisiones más adecuadas a cada situación, se requiere de información del entorno que alimente el sistema de procesamiento. De esta forma podrá obtener la solución más eficaz en cada momento concreto dependiendo de la situación.

Esto es fundamental, ya que éste no se puede ser un sistema cerrado obteniendo solo información del propio sistema. Para poder llegar a ser un vehículo totalmente autónomo, el sistema de control debe tener un flujo constante de información que provenga tanto del interior, como del entorno que lo rodea.

Para realizar este cometido, los coches autónomos cuentan con baterías de sensores de distintos tipos que le proporcionaran al sistema la información necesaria para determinar las magnitudes físicas del entorno que se precisen en cada momento.

Por esto, existe gran variedad de sensores. Sin embargo, en el campo de la automoción los que más se utilizan son los listados a continuación. [5]

Sensores de ultrasonidos

Se trata de sensores basados en la emisión y posterior recepción de ondas sonoras. Estos son los ultrasonidos, ondas sonoras de tan alta frecuencia, que no son perceptibles para nuestros oídos. Estas ondas se emiten desde el sensor colocado en el vehículo, que al llegar a objetos cercanos son reflejadas de vuelta.

Con esta tecnología se puede localizar los objetos cercanos y determinar con exactidud la distancia a la que se encuentran. Debido a su precisión a corto alcance, son muy utilizados en los bajos de los coches para la detectar objetos que al estar tan cerca quedan fuera del área de trabajo de otros sistemas. Generalmente suelen estar colocados en la parte delantera y trasera del vehículo.

Por todo esto, se suelen utilizar durante los estacionamientos, debido a las condiciones ideales que durante estas maniobras se presentan, movimientos lentos y con obstáculos muy cercanos.

Radar

Se trata de sensores análogos a los de ultrasonidos, pero con la diferencia de que estos usan ondas electromagnéticas en lugar de sonoras.

Estas tienen la ventaja de que su alcance es mucho mayor, llegando hasta los 200 metros en algunos casos. La limitación que hace que este sistema no se use frente a los ultrasonidos es que solo logra posicionar los cuerpos en el plano, no tiene percepción de la altura. Esto supone una gran limitación, ya que, aunque podemos saber que hay un objeto en una región del espacio no podemos tener la certeza de su forma y tamaño más que en una dimensión. Esto nos haría muy difícil distinguir con su empleo que clase de objeto estamos detectando.

Este sistema es más útil en situaciones exclusivamente de carretera, donde todos los cuerpos detectados son vehículos circulando en la misma dirección e importa más saber sus posiciones.

Lidar

Son sensores basados en el láser, de ahí sus siglas del inglés Light *Detection and Ranging*. Como los dos anteriores envía y recibe señales, en este caso de luz, mediante el uso de un láser. También tienen un alcance mucho mayor que el de los ultrasonidos, pero tiene la ventaja de que al trabajar con ondas de luz en el espectro de la luz ultra violeta, será prácticamente instantáneo y además nosotros no percibiremos nada a simple vista, ya que análogamente al caso de los ultrasonidos, no somos capaces de ver frecuencias de onda tan altas.

Estos sistemas suelen ser empleados para crear mapas del entorno cercano, generalmente en los 360 grados del coche, con lo que se obtiene una mayor percepción de los alrededores. Por esta razón, los sensores *LIDAR* se suelen colocar en la parte superior del coche donde puedan tener todo el rango de visión posible.

Otra de sus ventajas es que estos dispositivos tienen una mayor resolución de las imágenes captadas que el radar. Esto es debido a que la longitud de onda es mucho menor, con lo que las ondas pueden reflejarse en superficies más pequeñas y volver al sensor para ser detectadas. [6]

Sensores de imagen

Se trata de cámaras convencionales de captura de imágenes, son el sensor más utilizado por su versatilidad. Esto es debido a que no solo son capaces de detectar objetos, sino que también pueden captar su forma de manera precisa. Esto último es muy útil en el caso de la conducción, ya que pueden diferenciar señales que tengan prácticamente el mismo tamaño, pero distintas formas, como pueden ser las de peligro y ceda el paso. Además, son capaces de captar colores, característica muy importante para la conducción debido a los distintos significados de ellos en las señalesempleadas en la señalización de las vías. Esta característica permite también distinguir entre señales con la misma forma, pero distinta serigrafia, como es el caso de una señal de prohibido la circulación o una de limitación de velocidad.

Una desventaja que tienen es que un solo sensor por sí mismo no proporciona información sobre la distancia, solo nos da información en un plano bidimensional, el alto y el ancho. Esto contrasta con los sensores de radar, que también nos da una señal bidimensional, pero esta era de distancia y tamaño en horizontal.

Debido a las necesidades de este proyecto de reconocer peatones entre otros objetos que no lo son, y por la flexibilidad que ofrecen, se utilizaran en este proyecto los sensores de imagen o simplemente llamados cámaras.

Sin embargo, no se utilizará solo un dispositivo, sino que en su lugar se utilizaran dos colocadas de forma paralela y ambas apuntando en la dirección en la que se mueve el coche. La razón de utilizar dos en lugar de una es que de esta forma podremos tener acceso a la distancia de los peatones una vez hayamos tratado y procesado la información que nos proporcionen. Esto se explicará en más detalle en el desarrollo del proyecto, pero su funcionamiento es similar a como lo harían nuestros dos ojos, mediante diferencias entre las medidas entre ambas imágenes.

2.2 Los cinco niveles de la conducción autónoma

Tras esta pequeña introducción al concepto de conducción autónoma y a los principales sensores que se encargan de tomar información del entorno, se llega al siguiente punto crucial del capítulo de introducción. Se trata de los grados de automatización en la conducción autónoma.

Según la NHTSA, la National Highway Traffic Safety Administration, que pertenece al departamento de transporte de Estados Unidos, los vehículos autónomos se pueden clasificar en 6 niveles atendiendo a su autonomía. Esta es la clasificación más utilizada a nivel mundial, aunque

puede haber pequeñas variaciones entre los niveles contiguos, o agrupar algunos de los últimos niveles, según el organismo que los defina. [7]

Nivel 0, sin automatización. Esto es el coche con los sistemas justos y necesario, no tiene ningún tipo de ayuda a la conducción, todas las acciones que son necesarias para controlar el vehículo son realizadas en su totalidad por el conductor del mismo.

Nivel 1, asistencia a la conducción. El vehículo es controlado por el conductor, pero cuenta con sistemas de asistencia para el control de la dirección o la velocidad. El vehículo no tiene la capacidad de variar por si mimo estos factores, solo ayuda cuando el conductor realiza las acciones. Ejemplos de este grado de automatización podría ser el ABS para las frenadas o el control de crucero para la velocidad.

Nivel 2, automatización parcial. En este nivel de automatización, el vehículo es capaz de controlar de forma simultánea labores de direccionamiento y control de potencia del vehículo. Sin embargo, el conductor debe estar prestando atención al entorno en todo momento, para complementar al sistema siempre que fuera necesario. En este nivel, el automóvil sería capaz de, por ejemplo, conducir en un carril de forma autónoma, pero si el conductor necesitara cambiar de carril necesitaría tomar el control teniendo en cuenta el entorno para realizar la maniobra, tras lo cual el sistema podría retomar el control.

Nivel 3, automatización condicional. El conductor sigue siendo totalmente necesario, pero ahora el sistema de conducción puede realizar todos los aspectos de la conducción bajo unas circunstancias muy determinadas sin la intervención humana. Sin embargo, cuando estas no se dan el conductor debe retomar el control del vehículo de forma inmediata porque el sistema no está preparado para manejarlas.

Nivel 4, automatización alta. El automóvil ya es capaz de realizar una conducción completa en circunstancias normales, y no es necesaria la atención del conductor en estas. Solo durante eventualidades muy concretas podría llegar a ser necesaria la intervención humana. El vehículo es capaz de avisar y anticiparse para que el conductor retome el control del vehiculo, llegando hasta el punto de detener el vehiculo y aguradar a que el conductor resuelva la situación de conflicto. El conductor a pesar de no ser necesario puede retomar el control del vehículo en cualquier momento.

Nivel 5, automatización completa. Este nivel es la meta de la automatización. El sistema controla todos los sistemas y se puede adaptar a todas las posibles eventualidades del tráfico. No es necesaria la presencia ni el control en ningún momento, sin embargo, como en el caso anterior, siempre que se desee, se podrá recuperar el control manual.

Con esta clasificación, el objetivo claramente es llegar al nivel 5, y tener así la capacidad de delegar todas las funciones en el sistema de control del vehículo, y despreocuparse durante todo el trayecto.

Esto permitiría hacer uso del vehículo a personas que por diversos motivos como tener distintos impedimentos físicos no son aptos para una conducción segura, aumentando así la utilidad y accesibilidad de este medio de transporte aún más.

Además, también resultaría atractivo para el resto de usuarios, ya que haría más cómodos desplazamientos largos entre ciudades o tediosos por el interior de las mismas, aumentando a la vez la seguridad. Esto sería así puesto que la mayoría de los accidentes se producen por errores humanos, despistes, imprudencias, etc. Si somos capaces de eliminarlos, seguramente también habría algún accidente debido a algún error en los sistemas, pero el objetivo es hacer que fuera tan improbable que sea prácticamente imposible, y esto solo se puede alcanzar con la iteración, el desarrollo y la mejora de los sistemas. [8]

2.3 Futuro de la conducción autónoma

Los vehículos, son un bien que se ha extendido y estandarizado de forma muy rápida, solo en España había más de 32 millones de vehículos en el año 2016, y según a ACEA, la asociación europea de fabricantes de vehículos, solo en ese año se produjeron casi 80 millones de vehículos en todo el mundo. [9] [10]

Los vehículos forman parte de nuestro día a día, y desde que se crearon los primeros coches, no han parado de mejorar, añadiendo nuevas características que aumentan las posibilidades de los mismos, al mismo tiempo que se aumenta la seguridad.

Estos sistemas que mejorar la seguridad están implementados en la mayoría de los vehículos modernos. Son sistemas de asistencia a la conducción como la dirección asistida o sistemas de frenado de emergencia más eficientes como el ABS que ya los damos por sentados y que ayudan mucho durante la conducción.

Gracias a estos sistemas, se reduce en gran medida tanto los accidentes, como las consecuencias que estos traen consigo. La conducción autónoma es alcanzar el punto en el cual el factor humano no sea algo que provoque accidentes.

Actualmente existen vehículos autónomos con una gran capacidad de autonomía a la hora de circular, aunque la mayoría no son sistemas que se puedan adquirir de forma comercial, son por el momento prototipos empleados para desarrollar sistemas más complejos por las grandes marcas de automóviles. Tesla sin embargo, además de ser una empresa líder en vehículos completamente eléctricos tiene su principal objetivo en mejorar su piloto automático. Actualmente cuenta con modelos comerciales que tiene una capacidad de navegación autónoma muy elevada.

Muchas compañías en el ámbito de la automoción como Mercedes, se están sumando a desarrollar sus propios sistemas para la conducción autónoma, además de otras compañías que no estaban tan vinculadas a la automoción como Google o Apple. Todas están buscando innovar en este ámbito, para disponer de sistemas propios, lo que hace difícil seguir la pista a todos los progresos de cada empresa.

Lo que sí que se puede constatar es que cada vez más, se implantan sistemas automáticos para tareas muy concretas en muchos de los coches actuales. Entre estos sistemas podemos destacar ayudas a la conducción en autopista que ayudan a mantener el vehículo dentro del carril, sistemas de frenado de emergencia en ciudad si se detectan obstáculos muy cercanos al vehículo, o sistemas de aparcamiento que realizan todas las maniobras necesarias.

En estos últimos años hasta la actualidad la industria del automóvil está siendo objeto de un gran desarrollo, y sin duda lo será en los años que están por venir. Empresas que originalmente no estaban relacionadas con el mundo de la automoción y que estaban basadas en productos tecnológicos como Google o Nvidia se están uniendo a la carrera de los sistemas autónomos. Algunas de estas están centradas en crear los sistemas de procesamiento y cooperar con otras compañías para tener productos cada vez más complejo, pero otras apuntan mucho más alto y quieren llegar a presentar sus propios diseños completos para estos vehículos autónomos. Todos apuntando al futuro para traernos sistemas más capaces y seguros para convertirlos en una realidad.

CAPÍTULO 3: Análisis de la situación inicial

Este proyecto, tiene como objetivo principal detectar y caracterizar peatones que se encuentren en el entorno del vehículo. Esto se debe hacer haciendo uso exclusivamente de los sensores más extendidos y polivalentes, las cámaras de video.

Durante su funcionamiento, el coche nos ira transmitiendo las imágenes que se tendrán que utilizar en tiempo real para procesarlas y obtener de ellas toda la información que estamos buscando. En el caso de este proyecto, nos interesara saber el número de peatones en una imagen, la distancia de cada uno de ellos al vehículo, y su velocidad relativa al mismo.

3.1 Datos base del proyecto

Para realizar todas las pruebas y ejecuciones del programa, vamos a necesitar una base de datos en la que basarnos, un lugar en el que podamos tener las imágenes que se captan con el sistema que necesitamos. Ahí es el punto donde haremos uso del proyecto ***KITTI Vision Benchmark Suite***.

Se trata de un trabajo realizado *Karlsruhe Institute of technology*, el *KIT*, que pertenece a los centros alemanes de investigación. También es parte del desarrollo del proyecto el instituto tecnológico de Toyota en chicago, el *Toyota Technological Institute*.

El desarrollo de KITTI Vision Benchmark Suite se remonta al año 2015 cuando comenzó a desarrollarse, y desde ese momento no han parado de aumentar el progreso de sus trabajos. En su página disponen de mucha información de todos ellos, y además ponen a disposición de cualquiera que esté interesado la mayoría de los conjuntos de datos de los que han partido.

Esto es muy útil para el proyecto que llevamos a cabo, ya que podemos utilizar el sistema de cámaras estéreo que han implementado de forma física para realizar desarrollar el programa que satisfaga los problemas que estamos tratando.

El sistema desde el que se han tomado estos datos era en principio un coche normal. En su parte superior, como suele ser habitual en los sistemas para coches, se instaló el conjunto de sensores que capta toda la información y lo pasa a un sistema informático.

En la siguiente fotografía se pueden ver los sistemas de sensores situados en la parte superior del vehículo.

Fig. 3.1 Vehículo con sistemas de sensores en su parte superior

Aunque el sistema cuenta con un sistema de GPS y un láser que cubre los 360 grados del coche, no se van a utilizar sus datos. Sin embargo, si se emplearan el sistema de cámaras equipado, que consta de dos grupos de dos cámaras cada una. El motivo de esto es crear un sistema lo más completo posible haciendo uso del mínimo número de sensores que sea posible y a la vez que se trate de hacer que sean lo más polivalente y económicamente accesible que se pueda.

Estas cuatro cámaras están colocadas de forma paralela y todas enfocadas a la parte delantera del vehículo. Cada cámara en un grupo está separado 6 centímetros entre sí, y los dos grupos entre si están a una distancia de 54 centímetros. Estas distancias se detallan en la siguiente imagen y serán útiles más adelante cuando tratemos de determinar las distancias a los peatones.

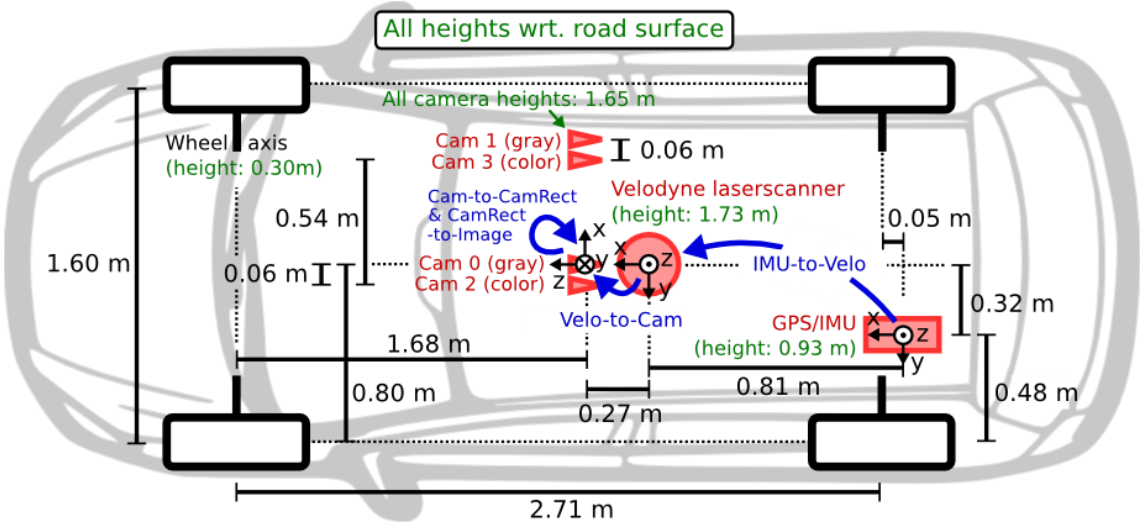

Fig. 3.2 Distribución de las cámaras sobre el vehículo del proyecto

En la propia página se puede descargar previa solicitud las imágenes que han obtenido durante sus trabajos. Estas son imágenes tomadas con las 4 cámaras durante recorridos por carreteras tanto en el ámbito urbano, como por autopistas. Los datos vienen en conjuntos de 180 imágenes correspondientes a distintos entornos por los que ha circulado.

Dado que lo que nosotros buscamos para el proyecto son ambientes con peatones, el set de imágenes que se buscaba es uno en el que el vehículo transitara por una zona con amplias aceras por las que en el momento de tomar las imágenes aparecieran suficientes peatones como para que nos resultara interesante. Al final el conjunto seleccionado para las pruebas es el número 0018 de la carpeta de *testing*, ya que era el conjunto que mejor cumplía estos requisitos.

Una vez ya tenemos la base de datos sobre la que vamos a trabajar, es el momento en el que se puede pasar a ver el software con el que se va a emplear para extraer toda la información necesaria de ellas.

3.2 Software, el lenguaje del proyecto

Este proyecto se basa prácticamente en su totalidad en el desarrollo de un software para el tratamiento de imágenes de las que se tratara de extraer toda la información necesaria. Para ello, como en todo software necesitaremos un lenguaje y si no queremos empezar desde cero unas librerías que nos aporten funcionalidades complejas de una forma más accesible.

En este caso, el lenguaje elegido es **C++**, esto se debe a que además de ser uno de los dos lenguajes que aceptan las librerías que vamos a utilizar, es un lenguaje con él ya que tenía algo de experiencia con él y por tanto, no partía también desde cero.

Se trata de una modificación del lenguaje C#, pero orientada objetos, lo cual lo hace más apropiado para trabajos complejos por su modularidad y agrupamiento de distintas funciones en clases diferenciadas para cada propósito.

Este lenguaje como todos los demás, tienen funcionalidades básicas con las que se puede implementar todos los programas que deseemos, siempre y cuando tengamos la habilidad de usarlas bien. Sin embargo, empezar un proyecto de este estilo desde cero no suele hacerse nunca. Para evitar esto están las librerías de funciones. Agrupaciones de clases y métodos que haciendo uso de las funcionalidades más básicas ofrecen capacidades mucho más complejas y que pueden ser incluidas en cualquier programa. [11]

Durante el desarrollo de la aplicación se han utilizado distintas librerías, pero la más importante, es **OpenCV.** El nombre viene del inglés *Open Source Computer Vision Library*. El paquete ha sido diseñado para el tratamiento de imágenes y obtención de información, justo lo necesario para la empresa que nos ocupa. Además, es de código abierto, lo que quiere decir que cualquiera puede hacer uso de ellas y seguir ampliándolas con nuevas posibilidades, cosa que se hace continuamente, convirtiéndolas en el mayor referente en la visión artificial.

Al tratarse de un sistema de funciones muy complejo es difícil controlar todas y cada una de ellas. Por suerte, cuenta con una gran cantidad de documentación online donde se puede consultar el funcionamiento de todos los métodos. En esta documentación se encuentran además algunos ejemplos que permitirán entender mejor el uso de las funciones en los lenguajes soportados.

De forma complementaria a esta documentación, para poder iniciarme en la visión artificial, he hecho uso de un curso online de la materia. Este curso se encuentra en la plataforma **EDX** es gratuito y se imparte por profesores de la universidad. El nombre del curso es "*Introducción a la visión por computador: desarrollo de aplicaciones con OpenCV"* y es un buen método para tener una primera idea general de su funcionamiento y de la visión artificial en general. [12]

3.3 Hardware empleado, JetsonTX2

Una vez ya se tienen los datos con los que vamos a trabajar, y la base principal del software con la que se van a tratar, aún queda saber el hardware que se va a emplear, el elemento físico en el que va a ocurrir todo el procesamiento.

Aquí es donde entra la compañía **Nvidia**, con su producto **JetsonTX2.** Este es un kit de desarrollo para sistemas autónomos. Es un pequeño ordenador, pero optimizado para labores de cálculo y procesamiento de los ámbitos de la visión e inteligencia artificial. Hace uso de la tecnología de las tarjetas gráficas de Nvidia para proporcionar un buen rendimiento en un sistema muy pequeño y eficiente. Este equipo fue prestado por la universidad para poder realizar el trabajo cuenta con todo tipo de conexiones e incluso incluye una cámara para poder probar aplicaciones de forma rápida sin necesidad de conectar otra externa. [13]

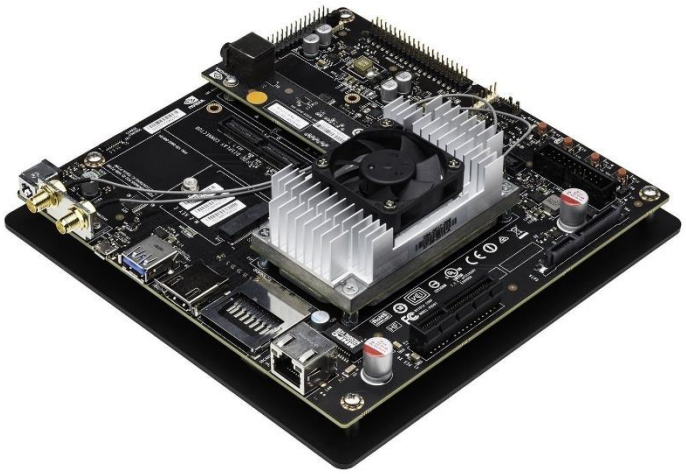

Fig. 3.3 Placa JetsonTX2 de Nvidia

El sistema como ya se ha mencionado es un pequeño, pero completo ordenador. Sin embargo, el sistema operativo que lo controla no es el habitual Windows al que estamos acostumbrados encontrar en la mayoría de dispositivos. Por el contrario, trae **Ubuntu**, un sistema operativo de código abierto que forma parte de la familia de **Linux**.

Ambos son sistemas operativos, pero Ubuntu está más orientado a la creación de aplicaciones, ya que dispone de muchas más opciones en el sistema para hacer posible instalar programas que de otra forma no sería posible. Prácticamente todo se puede modificar, que es la gran ventaja de que sea un entorno abierto, lo que contrasta con un sistema más extendido como es Windows, que tiene sistemas más intuitivos, pero mucho más rígidos en cuanto el abanico de posibilidades.

Dentro de este sistema operativo, no ha sido necesario el empleo de ningún programa especial para la programación. Gracias a la versatilidad del sistema operativo, basta con un block de notas para, usando la consola del sistema, poder compilar el programa y ejecutarlo. Con esto, lo que se pretendía es reducir en la medida de lo posible la curva de aprendizaje de un sistema con el que no se está familiarizado, minimizando también el tiempo dedicado a la puesta a punto de los sistemas previos al trabajo propiamente dicho.

CAPÍTULO 4: Diseño de la solución

En este punto ya se conoce todo el hardware con el que se va a trabajar, y el software base. Ahora lo más importante es tener un plan para ver qué es lo que tenemos que conseguir y como pretendemos alcanzarlo.

De esta forma, el diseño de la solución del problema se ha separado en cinco fases. Cada una de ellas estará centrada en partes muy diferenciadas. El objetivo de esta organización es tener un sistema modular. Esto nos trae muchas ventajas tanto a la hora de diseñar el código, como a la de reutilizarlo en futuros trabajos.

En primer lugar, nos permite desarrollar el programa de forma separada, dándonos la posibilidad de alternar el trabajo de programación entre cada una de ellas de forma independiente. Esto resulta muy útil cuando lo que se requiere es tiempo para ver cómo solucionar un problema complejo mientras se puede seguir avanzando con otras partes menos exigentes.

La segunda ventaja como ya se ha dicho es que es más fácil de cara a trabajos futuros, ya que será más sencillo extraer alguno de los módulos reemplazarlo, mejorarlo o suprimirlo.

El único problema que puede tener es que el código es mucho menos compacto, y posiblemente algo menos eficiente. Sin embargo, considero que para la naturaleza y objetivo de este proyecto resultara mucho mejor el empleo de esta estructura modular.

Los siguientes subapartados van a ser las propias divisiones en fases, donde se explicarán en más detalle cada una de las soluciones adoptadas. Estas se encuentran representadas en el diagrama inferior, donde el primer y último recuadro representan tanto las entradas como las salidas del sistema, y las cuatro fases en azul representan todos los pasos intermedios que se han tenido que dar para llevar a cabo la transformación.

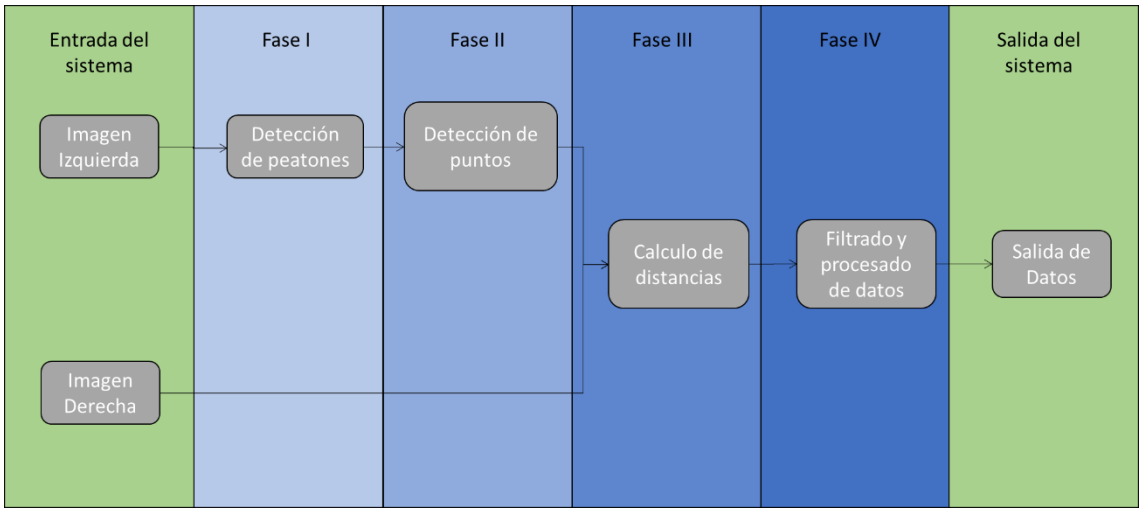

Fig. 4.1 Diagrama de funcionamiento del programa

4.1 Entrada de datos del sistema

Como ya se ha mencionado anteriormente, la entrada de datos del sistema se realiza mediante las imágenes de la base de datos de Kitty. Estas imágenes se han descargado y a cada ejecución del programa, lo primero que debe hacer es leerlas del disco duro para tenerlas en memoria. Este paso es fundamental, ya que, al cargar todos los datos de una vez, no tenemos que volver a acceder al disco duro, que es un proceso que es bastante lento.

Las imágenes que se van a cargar son las imágenes en color, que, aunque no son las que más se requieren por las funciones de procesamiento, se pueden modificar fácilmente. La principal ventaja de hacerlo con imágenes en color es que podremos analizar los resultados de una forma más sencilla posteriormente.

En el caso de que como fuente de datos se tuviera un video, el procedimiento seria el mismo, solo habría que modificar ligeramente las funciones de lectura para que pudieran acceder al video.

Cuando se quisiera implementar este programa y ejecutarlo en tiempo real con los datos que están obteniendo las cámaras se podría hacer de dos formas, en el propio código acceder a las imágenes de cámaras directamente, o si el sistema fuera muy complejo haciendo uso de otros programas externos disponibles en Ubuntu como es ROS. Un sistema para la integración de sistemas que permitiría hacer accesible la información de sistemas distintos en un único sistema.

4.2 Fase I. La detección de peatones

El primer paso que se debe dar es detectar los peatones presentes. Este es el primer paso lógico a seguir dado que queremos hacer uso de las propiedades de los pares de imágenes, pero no de todo su conjunto. Primero tenemos que saber en qué zona de la imagen es donde tenemos que extraer esa información puntual.

Para lograr esta identificación, se ha hecho uso de otro proyecto previo **Jetson-Inference**. Este es un programa diseñado para el reconocimiento de elementos dentro de imágenes. Se basa en redes de inteligencia artificial entrenadas para reconocer distintos objetos.

Este es un software que se pone a disposición de todos los usuarios en la red por la compañía Nvidia. Esta diseñado especialmente para el hardware que se emplea y sirve como base para aquellas personas que quieran realizar otros proyectos en los ámbitos de la visión artificial.

La documentación de la aplicación está enfocada a la creación de redes de identificación propias. Esto nos permitiría mediante el procedimiento que se enseña en la página, el uso de una base de datos suficientemente grande y el hardware adecuado, entrenar al sistema para que sea capaz de reconocer cualquier elemento en una imagen. En nuestro caso serían los peatones, pero se podría entrenar para cualquier otra finalidad como, por ejemplo, distinguir señales de tráfico.

Por suerte, no ha sido necesario entrenar una red desde cero ya que también ponen a disposición unas que han sido previamente entrenadas para reconocer distintos elementos como vehículos o peatones, que es lo que perseguimos en este proyecto.

Este software, según su licencia de uso, nos indica que somos libres de reutilizar, o realizar las modificaciones que necesitemos sin ningún tipo de restricción de uso o distribución al respecto.

Toda la información sobre el programa puede encontrarse en su página de *GitHub***.** [14]

Por todas sus características, junto con que ha sido diseñado específicamente para el equipo utilizado, ha sido el software seleccionado para servir de base para comenzar el sistema para el vehículo autónomo.

4.2.1 Segnet, la segmentación semántica

Una vez instalado el proyecto Jetson-Inference en el equipo JetsonTX2, podemos acceder a distintos métodos de reconocimiento, entre los que destacan dos, la segmentación semántica y la detección de objetos. Ambos a priori pueden detectar entre otros cuerpos a los peatones, por lo que eran apropiados.

Siendo así, se optó por comenzar a utilizar la segmentación semántica, por su mayor precisión.

La **segmentación semántica** es un procedimiento por el cual todos los pixeles de una imagen se clasifican en las categorías que tenga aprendida la red de identificacion. De esta forma, la salida clásica de estos procedimientos es la misma imagen, pero con una capa superpuesta que nos califica las regiones de la imagen en las categorías disponibles, según los objetos que se detecten en ellas.

Un ejemplo de esto es la imagen inferior, una imagen de muestra de la documentación del programa. Podemos ver como todo los pixeles que pertenecen a vehículos se les ha aplicado una máscara de color azul, los peatones los ha identificado con el color rojo, las señales de amarillo, y otros colores para el resto de categorías.

Fig. 4.2 Imagen segmentada modelo

La segmentación como cualquier método de inteligencia artificial no siempre nos dará los resultados correctos, pero se entrenan con grandes bases de datos para ofrecer porcentajes de confianza muy altos y así tener menos error.

La principal ventaja de este método frente a la detección de objetos es que la salida nos la ofrece pixel a pixel, y no como un área con una forma fija donde se detecta el objeto. Esto es muy útil para no mezclar la información que queremos analizar con la del fondo.

Para nuestro proyecto, solo necesitamos identificar las personas, lo que el programa identifica en color rojo, el resto de información es irrelevante para el objetivo marcado. Por tanto, para no cargar al sistema calculando información innecesaria, empeorando el rendimiento, deberíamos hacer que solo identifique las partes de la imagen en las que se presenten peatones, y deje sin marcar el resto de la imagen.

El código que debemos ejecutar para realizar la segmentación es el que viene bajo el nombre de *segnet_console*. Este nos permitirá cargar una imagen de la memoria del dispositivo, realizar la segmentación y obtener una imagen resultado con los resultados, todo en una estructuras del programa lineal, sin la opción de volver a ejecutarlo sin antes volver a abrir la aplicación.

Cuando lo ejecutamos, lo que hace es buscar todas las estructuras de las clases en las que ha sido entrenado. El que está a nuestra disposición en la página esta entrenado para detectar 21 clases distintas. Estas se pueden encontrar únicamente en la programación del algoritmo y no en la documentación, por lo que no fue trivial su identificación. Concretamente lo podemos visualizar en un archivo llamado *cityscapes-labels* dentro de las carpetas que hay que instalar para hacer funcionar el programa, pero se han extraído y se muestran en la tabla inferior para una presentación más clara.

Clase	Nombre	Clase	Nombre
00	Void	11	Pole
01	Dynamic	12	Traffic light
02	Ground	13	Traffic sign
03	Road	14	Vegetation
04	Sidewalk	15	Terrain
05	Parking	16	Sky
06	Building	17	Person
07	Wall	18	Car
08	Fence	19	Truck
09	Guard rail	20	Cycle
10	Bridge tunnel		

Tabla 1 Clases disponibles para Segnet

Como se puede ver, la única categoría que nos interesa es la 17, *Person*. Así la primera modificación del código debe ser hacer que si no es de la clase 17 la ignore y no lo notifique a la imagen salida del programa.

Además, no nos interesa tener la imagen original con una superposición de los resultados. Eso solo es útil para la visualización para nosotros. El programa solo necesita saber las regiones de la imagen donde tiene la información de una forma inequívoca. Por tanto, la otra modificación fundamental que se hizo a la salida del programa es que esta solo sea la capa que superpone a la imagen, lo que en el código se denomina como *overlay*.

La forma más sencilla de hacer esto es aplicar esta capa de información a una imagen con el mismo tamaño que la imagen original, pero con todos los pixeles vacíos, sin información, es decir, una imagen en "negro".

También sería necesario cambiar el color del peatón detectado a blanco para hacerlo más fácil, esto habría que hacerlo modificando el archivo donde se especifican el color de cada clase. El archivo no se encuentra especificado tampoco en la documentación, pero tras buscar por los archivos del programa se puede encontrar en un documento de texto llamado *cityscapes-deploy-colors*. En el podemos ajustar los valores RGB para tener cada clase en el color deseado.

Con estas pequeñas modificaciones hemos conseguido que de una imagen nos devuelva otra en la que todos los pixeles con el color blanco son aquellas en las que el sistema ha detectado que hay peatones, y el resto de la imagen está vacía.

Estas transformaciones se pueden apreciar de una forma mucho más visual con las siguientes imágenes. Pertenecen a una de las imágenes de muestra que nos proporciona el proyecto Jetson Inference, *"peds-001.jpg"*.

Esta primera imagen corresponde a la imagen original sin ninguna modificación.

Fig. 4.3 Imagen de muestra peds-001.jpg

La siguiente imagen es la salida que se obtiene del programa sin realizarle ninguna modificación si le introducimos la imagen anterior para que la procese.

Fig. 4.4 Imagen peds-001.jpg segmentada

Por último, la siguiente imagen es el resultado de procesar la imagen original con el sistema modificado que se ha descrito, donde solo aparecen en blanco las regiones clasificadas como peatones.

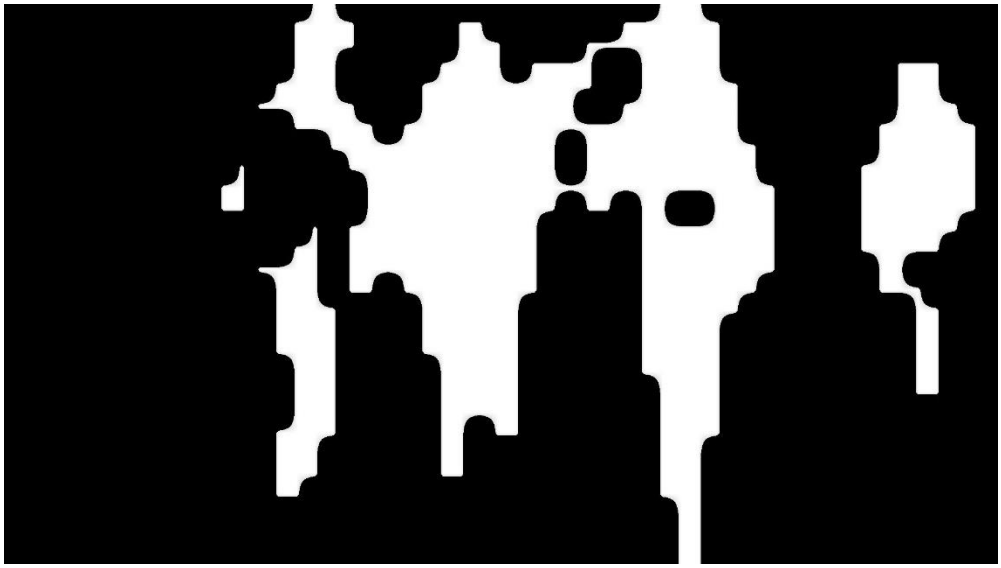

Fig. 4.5 Imagen peds-001.jpg segmentada y modificada

Con esto pudiera parecer que ya estaría solucionada la parte de detección de peatones, sin embargo, tras tener el código listo y funcionando, se presentó uno de los inconvenientes del diseño modular. La segmentación no se realizaba correctamente en las imágenes de la base de datos del proyecto Kitty.

Ninguna imagen en la que aparecían personas el programa las detectaba, las imágenes de salida estaban todas vacías, sin ningún objeto detectado. De hecho, tras volver a dejar el programa en su versión original, no realizaba ninguna segmentación a estas imágenes, por lo que se puede asegurar que no es un fallo inducido por las modificaciones realizadas.

Esto sin duda es un inconveniente, y dado que se estaba siguiendo la estructura modular, no fue hasta el momento de unir todas las partes cuando se detectó el problema.

La causa de esto no está del todo clara, ya que en la documentación no hay ninguna especificación sobre las imágenes de entrada. Sin embargo, todo apunta a que las imágenes de la base de datos no son muy nítidas, son algo más "borrosas" de lo normal, por lo que no puede realizar la ejecución de forma correcta.

Aunque este método no haya funcionado, habría sido una opción que habría aportado un mejor resultado final, con lo que ha merecido la pena probar, y si en un futuro se usa otra base de datos que si lo acepte, seguramente terminaría dando resultados más precisos como medio principal de detección o como método secundario para el refinamiento de la detección de peatones.

4.2.2 Detectnet, la detección de objetos

Por suerte Jetson Inference cuenta con otro método adicional para detectar peatones, *detectnet*. Este sistema a diferencia del anterior no detecta los pixeles en los que hay peatones, sino que lo que hace es delimitar unas zonas en las que se encuentra el objeto detectado.

Estas zonas cuadradas o rectangulares, se denominan boundingboxes, y su forma es un paralelogramo que modificara su tamaño en cada caso dependiendo de la forma que mejor se adapte al objeto que pretende abarcar. Sin embargo, por mucho que se quiera acercar, con este método siempre quedarán zonas más o menos grandes dentro de la región delimitada que no sean parte del cuerpo, serán parte del fondo, por lo que requerirá de algún tipo de filtro que nos permita eliminar los datos de estas regiones que no nos aporten información valida.

Como en el caso anterior con la segmentación, detectnet es capaz de detectar más clases además de peatones, por lo que tendremos que seguir un procedimiento similar deshabilitando la detección de todo aquello que no nos resulte útil para la aplicación que estamos creando.

En este caso los tipos de entidades que se pueden detectar no están en un archivo aparte, sino que están dentro de la definición de las clases del programa, más concretamente dentro de **detectNet.h**, fichero donde están todas las definiciones para este segundo método.

Esta vez las clases que tiene no son tan enfocadas a la conducción, ya que tiene clases mucho más variadas en temática. Todas ellas se encuentran recogidas en la siguiente tabla junto a una pequeña descripción, que nuevamente no se encuentran mencionadas en la documentación, por lo que su presentación en la siguiente tabla resulta muy conveniente.

Tipo	Descripción
COCO_AIRPLANE	Detección de aviones
COCO_BOTTLE	Detección de botellas
COCO_CHAIR	Detección de sillas
COCO_DOG	Detección de perros
FACENET	Detección de caras humanas
PEDNET	Detección de peatones
PEDNET_MULTI	Detección de grupos de personas y de equipajes

Tabla 2 Clases disponibles para detectnet

De estas siete categorías solo tres están referidas a personas, a pesar de eso, solo usaremos las dos últimas. El motivo de esto es que no necesitamos detectar caras, eso podría llevar a errores innecesarios al sistema y además no nos daría ninguna información útil adicional. Por tanto, dejaremos las detecciones de peatones individuales y las de conjuntos de personas, porque estos si están sujetos a ser detectados para cumplir con el objetivo de este proyecto.

Al igual que no caso anterior, el sistema original al darle una imagen nos la modificara y tendremos la misma imagen de entrada pero con los recuadros de las boundingboxes detectadas superpuestas. Esto no lo podremos utilizar directamente en el resto del programa, sino que tendremos que transformar el formato de la salida de una forma muy similar a como hicimos durante la segmentación.

En este caso lo que buscamos es que nos devuelva una imagen vacía, pero con los pixeles donde se encuentras lo peatones “activados”, la forma más sencilla de hacer eso es que los ponga en blanco, lo que nos dará un mayor contraste con el fondo negro por si hiciera falta revisarlo de forma visual. Además, en los valores numéricos de los pixeles es lo que más sentido tiene, ya que el color negro se representa como 0 0 0 en cada uno de los tres colores RGB (rojo, verde y azul), y el blanco por el contrario tiene el valor máximo en cada uno de ellos, 255.

Este formato además de ser fácil de comprobar de forma visual, nos será útil para los pasos posteriores en los que se utilizará esta información como base para otros procesos.

Un ejemplo de la transformación que sigue la imagen a su paso por el programa se encuentra en las siguientes imágenes. Se corresponden con las trasformaciones que se han descrito para la primera imagen de la base de imágenes que se va a utilizar.

La primera siendo la salida original del programa y la segunda la salida una vez realizada la modificación que se ha descrito para poder ser utilizada en procesos posteriores.

Fig. 4.6 Primera imagen del set tras su paso por el proceso detectnet

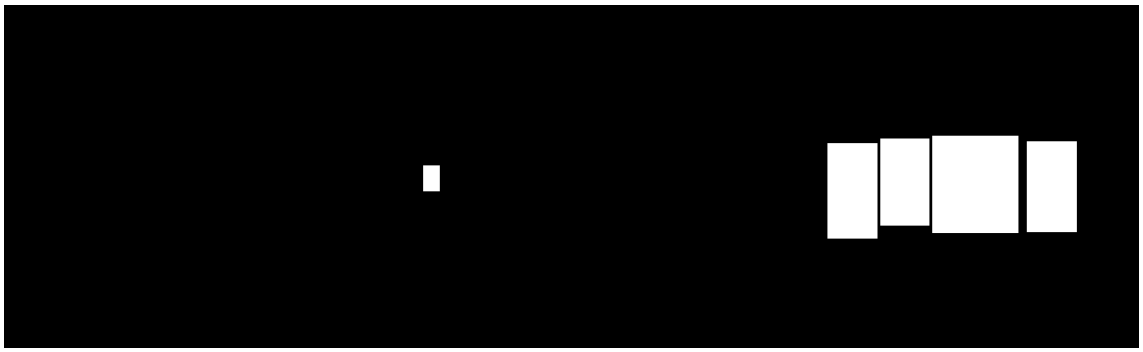

Fig. 4.7 Resultado de la modificación de la primea imagen

Se puede ver como los peatones de la derecha los detecta todos perfectamente agrupando dos de ellos gracias a tener la clase PEDNET_MULTI para conjuntos de peatones, que de otra forma no habrían sido reconocidos. Además, reconoce a un peatón al otro lado de la calzada que está bastante alejado.

Aunque es cierto que hay más peatones que no se detectan, es porque están parcialmente tapados como es el caso de los que hay a la izquierda o bien están en unas condiciones de iluminación que hacen muy difícil localizarlos como es el caso del grupo de peatones de la acera derecha que están muy alejados. Los resultados se explicarán con más detalle en el capítulo de *Análisis de resultados*.

4.3 Fase II. Asignación y seguimiento de puntos críticos.

El objetivo de esta segunda fase es determinar puntos de interés en las regiones de la imagen que vamos a estudiar.

En la primera fase se ha reducido la totalidad de la imagen a unas pocas regiones rectangulares en las que se sabe esta la información de la que queremos obtener datos. Aun así, la región sigue siendo muy amplia, por lo que hay que seguir reduciendo los lugares con información potencial hasta llegar a los pixeles concretos. Esta es la única manera que tenemos de extraer la información por el momento, con los valores de color e intensidad de cada pixel, por lo que tenemos que afinar hasta obtener esa precisión.

4.3.1 Método de detección de esquinas

Lo primero que tenemos que hacer es determinar qué puntos de estas regiones son realmente importantes y podrían tener mayor relevancia. Para eso utilizaremos un algoritmo de detección de esquinas.

Este algoritmo tiene su origen en 1988, cuando Chris Harris y Mike Stephens desarrollaron un algoritmo capaz de detectar bordes en de objetos en imágenes. El algoritmo tiene el nombre de ***Harris Corner Detector***, o el detector de esquinas de Harris en español.

Este algoritmo recorre toda la imagen calculando la variación de intensidad de cada punto si se realiza desde él un desplazamiento en otras direcciones. Este algoritmo trabaja con imágenes en blanco y negro, lo que facilita el cálculo de estas variaciones ya que solo tiene un valor de intensidad por cada pixel de la imagen. La salida nos devolverá otra imagen también en escala de grises en la que destacaran los puntos en los que exista un mayor cambio en su entorno, las esquinas. [15]

La fórmula de este algoritmo se muestra a continuación.

$$f(x,y) = \sum_{x,y} (I_{(x,y)} - I_{(x+\Delta x,\ y+\Delta y)})^2$$

Esta realiza un sumatorio en cada punto de la región analizada de coordenadas x, y de la diferencia de cuadrados del valor del punto con su entorno. El resultado de esta función nos dará valores mas altos en aquellos puntos donde la intensidad varié mas en su entorno, los bordes de las imágenes.

Este fue uno de los primeros algoritmos que daban unos resultados fiables. Sin embargo, el que vamos a utilizar es una modificación de **Shi-Tomasi** en el que se cambiaba la ponderación de las diferencias de intensidad con lo que se obtienen mejores resultados.

La función que tienen las librerías OpenCV que implementa este algoritmo es ***goodFeaturesToTrack***. La función necesitará que se le introduzcan unos parámetros específicos, a continuación, se detallan cuáles son y su motivo de elección.

- El primero de estos parámetros es la imagen a al que se va a realizar el algoritmo. Esta imagen debe estar en blanco y negro, y en formato *Mat*, el formato estándar designado por las OpenCV para trabajar imágenes.

- El segundo es la salida que nos devolverá su ejecución. En este caso se trata de un vector, o array, de datos de dos dimensiones. Cada uno de los elementos que nos devuelve serán los puntos detectados y se denotarán mediante sus coordenadas X e Y de la posición de cada pixel dentro de la imagen de entrada.

- El tercer parámetro es otra entrada que marcará el número máximo de puntos que podríamos obtener a la salida. En el programa se ha fijado a 400, que, aunque es un número bastante grande se reducirá con el siguiente parámetro y solo servirá para tener un límite superior y así evitar cargar demasiado al sistema en caso de que el otro parámetro no lo limite adecuadamente.

- El próximo parámetro que se debe de introducir es el nivel de calidad de los puntos. Es un numero decimal entre 0 y 1 que nos fijara el límite inferior para el cual un punto se rechazara si no lo alcanza. Si el mejor punto de todos tiene una calidad de 100 según lo obtenido con el algoritmo, y nosotros ponemos el nivel de calidad a 0.1, eso quiere decir que cualquier punto que sea menor al 10% del mejor será rechazado. En el programa tenemos un nivel de 0.01, lo que equivale al 1% para así reducir el número de puntos desde 400 pero teniendo una gran cantidad para mejorar así los resultados.

- El siguiente parámetro es la distancia mínima entre puntos. Con esto indicamos como de juntos pueden estar los puntos que obtendremos a la salida, medido en pixeles. Para este valor nos interesa que sea relativamente bajo para así poder tener mayor cantidad de puntos en un mismo espacio. Mediante prueba y error con distintos valores se ha llegado hasta un valor final de 6. Este resulta ser un número que funciona muy bien ya que es lo suficientemente pequeño como para maximizar la cantidad de puntos que podemos meter sin hacer que se superpongan demasiado en zonas con gran volumen de puntos.

- El ultimo parámetro que se necesita introducir es la máscara. Este parámetro opcional es otra imagen que debe tener el mismo tamaño que la principal, y estar en blanco y negro, ya que solo acepta valores de 0 a 255. Esta imagen será la que se ha obtenido de la modificación de la segmentación. Lo que hace esta máscara es determinar las regiones en las que se debe quiere ejecutar el algoritmo. Si estos datos no se pasan, se ejecuta en toda la imagen, pero para agilizar el proceso se incluirá y de esta forma el algoritmo solo se ejecutará en las regiones delimitadas con el valor 255, el color blanco.

El resto de parámetros se refieren a modificaciones del propio algoritmo, pero que con sus valores por defecto da unos valores lo suficientemente buenos como para no necesitar modificarlos en esta aplicación. Todos ellos están descritos en la documentación de las OpenCV. [16]

Un ejemplo de la salida de este *goodFeaturesToTrack* con los parámetros que se han comentado es la siguiente imagen. Esta es de nuevo la primera imagen del set de la base que se ha empleado.

Fig. 4.8 Primera imagen tras aplicar GoodFeaturesToTrack

En esta imagen podemos ver marcados todos los puntos detectados en verde. Esto se ha hecho únicamente para poder visualizar los resultados creando y aplicando una nueva capa en la que colocar círculos superpuestos a la imagen original en las coordenadas que nos indicó el algoritmo.

Si la comparamos con la imagen en la que se detectaban los peatones, la *figura 10*, se puede ver como la totalidad de los puntos están dentro de las regiones delimitadas, esto es gracias a la máscara que creamos. También podemos apreciar cómo, aunque el número de puntos no se acerca al límite, son muy abundantes en todos los peatones que están más cerca y que estos se concentran en su mayoría en sus contornos.

También podemos ver cómo, aunque la mayoría de los puntos están en los peatones, hay alguno que está dentro del recuadro, pero está en el fondo. Esto se tendrá que solucionar en el *módulo del filtrado*, ya que el algoritmo no tiene forma de distinguir entre objetos, solo puede encontrar bordes en sus regiones de trabajo.

4.3.2 Método de flujo óptico

Con esto, ya queda solucionada la cuestión de la selección de los puntos críticos de la imagen. Sin embargo, estos no van a ser estacionarios, ya que tanto el peatón como el coche generalmente estarán en movimiento. Para esto necesitaremos hacer uso de otra función distinta, **calcOpticalFlowPyrLK**.

Se trata de un algoritmo que nos permite detectar puntos en una imagen posterior a una en la que se conocen sus coordenadas, en otras palabras, realiza un seguimiento de puntos entre imágenes consecutivas. El algoritmo debe ser capaz de encontrarlos en la imagen siguiente y nos debe proporcionar sus nuevas coordenadas. De esta forma no se tendrá que aplicar siempre el método de detección de esquinas, basta con hacerlo una vez para inicializar y después hacer el seguimiento de los puntos con este método de flujo óptico.

Está función está basada en el algoritmo de **lucas-kanade** para flujo óptico, pero con la modificación piramidal de **Jean-Yves Bouguet**. Esta aporta una significativa mejora al algoritmo de búsqueda de los puntos, ya que utiliza una estructura piramidal. Esto, según el autor, consiste en aplicar el algoritmo original a distintas copias de la imagen. Cada una de estas copias son como los estratos de la pirámide, cada vez más pequeñas, de ahí su nombre. Con todas estas se puede extraer la información de movimiento y coordenadas de forma mucho más precisa debido a la comparación con las diferentes modificaciones de las imágenes que se procesan.

Según explica el propio autor estas pirámides no suele tener más de 4 estratos, ya que no se obtienen mejores resultados con más niveles. Además, la modificación piramidal “es capaz de manejar mayores desplazamientos de los pixeles” que la versión sin la modificación del algoritmo. [17]

La función de las OpenCV comparará dos imágenes, la primera en la que tenemos los puntos y sus coordenadas con la segunda en la que queremos saber a qué posiciones se han desplazado esos puntos. Los parámetros de esta se detallan a continuación.

- Las dos primeras entradas que necesita esta función son las dos imágenes, la primera debe ser a la que pertenecen los puntos iniciales de los que conocemos sus coordenadas y la segunda en la que se quiere averiguar a dónde se han desplazado. Ambas tienen que estar en formato *COLOR_BGR2GRAY*, que es la manera que tiene OpenCV de denominar a las imágenes en blanco y negro. Estas se pueden obtener fácilmente de las imágenes que nos ofrece Kitty database con una rápida conversión de formatos.

- Los dos parámetros siguientes son los arrays de puntos. El primero será una entrada, y contendrá las coordenadas de la imagen original, mientras que el segundo será la salida del proceso, donde nos devolverá las coordenadas en la nueva imagen.

- El siguiente es *status*, otro vector que devolverá información del proceso. Esta será binaria, ya que solo nos devolverá unos y ceros. Cada uno de los puntos de entrada tendrá su indicador de *status*. La información se refiere a si se ha podido calcular el punto asociado a cada dato que se ha proporcionado a la entrada de la función, o si por el contrario el punto, y por tanto la información, se ha perdido en la transición entre imágenes.

  Un 1 nos indicará que se ha realizado de forma correcta. Esto nos será útil para llevar la cuenta de cuantos puntos tenemos en cada momento, ya que, aunque no puede aumentar desde que los detectamos con *goodFeaturesToTrack*, si puede disminuir si se pierde alguno. Generalmente se debe a que se salgan de la imagen o se interponga algún cuerpo y no siga apareciendo en la imagen.

- La última salida que tenemos es *err*. Esta nos devolverá el tipo de error que se ha producido en alguno de los puntos si es que tiene alguno. Para la aplicación que estamos buscando esto no nos será de utilidad, ya que si perdemos un dato ya no vamos a poder recuperarlo hasta la nueva inicialización.

El resto de variables no hará falta modificarlas, ya que sirven para cambiar parámetros del algoritmo, pero que tampoco se emplearán para esta aplicación. La información de tallada de los otros parámetros también la podemos encontrar en la documentación online de las librerías. [18]

La aplicación de esta función debe estar condicionada a tener una imagen anterior en la que se hayan calculado los puntos. Por lo tanto, la imagen siguiente de ejemplo no se trata de la primera del set, si no de la segunda.

Fig. 4.9 Segunda imagen tras aplicar CalcOpticalFlowPyrLK

Nuevamente los puntos están marcados en verde y en esta imagen se muestran los correspondientes a esa misma imagen. Ahora hay un elemento adicional, las líneas rojas. Estas representan el desplazamiento de cada punto respecto a la posición que ocupaban en la imagen anterior.

Se puede ver como los peatones que están más cerca tienen, en general, movimientos más grandes que los que se encuentra más lejos, que apenas son perceptible. Esto se debe a la posición relativa de ambos, al estar mucho más cerca, recorrerán una mayor parte de la imagen que el peatón que está más alejado, aunque ambos grupos tengan velocidades relativas similares.

Con esto ya se tiene acceso a los medios para obtener puntos de la imagen de los que obtener la información de posición. Habiendo aplicado estas funciones ahora tenemos dos datos fundamentales con los que trabajar, la posición dentro de la imagen, que nos servirá para seguir obteniendo información en zonas muy concretas, y su movimiento entre las sucesivas imágenes, que nos proporcionará la capacidad de calcular desplazamientos.

4.4 Fase III. Cálculo de distancias.

Esta tercera fase será la última en la que extraigamos nueva información de las imágenes de forma directa. Esta información que se necesita para terminar de ubicar los peatones es la distancia a la que se encuentran. Para ello haremos uso de la segunda imagen que se toma en cada momento desde la cámara paralela.

Hasta el momento la información que teníamos la podíamos sacar de cualquiera de las dos imágenes indistintamente ya que son prácticamente iguales en cuanto a información. La única diferencia es que cada una de ellas nos dará ligeramente más visión del lado en el que están colocadas y un poco menos del contrario. Sin embargo, para calcular las distancias necesitaremos hacer uso de ambas de forma simultánea.

A este proceso se le llama ***stereo maching***, no hay una traducción muy exacta al español, pero sería algo así como visión estereoscópica, ya que utiliza dos canales para producir la información. También se le conoce como ***disparity mapping***, o mapeado de disparidad en castellano.

Lo primero que hace esta técnica es alinear las imágenes de forma vertical. De esta forma dos pixeles que representen el mismo punto de la realidad estarán a la mima altura o coordenada Y. Esta primera fase de rectificado es útil para eliminar pequeñas imprecisiones que tenga la captura de la imagen para asegurar que el resto del algoritmo se ejecute de forma correcta.

La segunda parte de este algoritmo consiste en dividir la imagen filas de 1 pixel de alto. Esto se les hace a ambas imágenes para poder comparar estas regiones con menos cantidad de información eliminando así posibles errores.

Después esas dos regiones se recorrerán hasta encontrar coincidencias en ambas. Para cada punto que se encuentre en las dos secciones se medirán las distancias entre las coordenadas de cada una y este será el dato que forme el mapa de disparidad.

Debido a las propiedades de las cámaras y de la perspectiva en general, los objetos que más cerca estén del objetivo tendrán una mayor disparidad que aquellos que se encuentre más lejos. Esto se debe a que, aunque dos objetos se desplacen la misma distancia, el objeto que esté más cerca del objetivo habrá recorrido un mayor ángulo respecto a la cámara. Por esto, al proyectarlo sobre la imagen que se genera, habrá una mayor distancia en el movimiento del objeto cercano que del lejano.

Esto se puede visualizar mejor con un ejemplo gráfico. En él se ha representado el campo de visión de una cámara y dos objetos, uno más cercano y otro más lejano. Ambos se desplazan la misma distancia, pero al proyectarse, el cercano parece que se ha movido más debido a que su movimiento abarca un ángulo mayor.

En nuestro caso no se trata de que el objeto se mueva en una misma toma, sino que las cámaras al estar separadas una distancia en horizontal, se produce el mimo efecto que si se hubiera desplazado toda la escena.

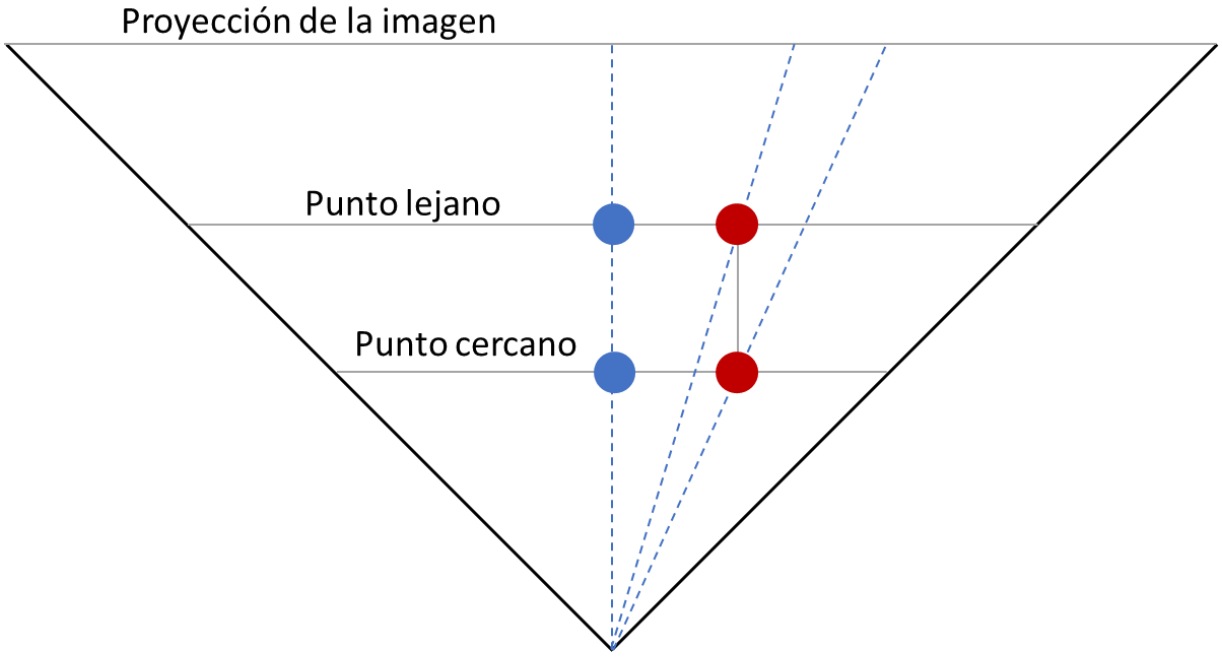

Fig. 4.10 Representación de proyecciones de objetos en movimiento a distintas distancias

Si aplicamos este proceso de cálculo de distancias a todos los puntos de ambas imágenes obtendremos el mapa de disparidad de la escena. Este mapa contendrá todos los valores de distancia entre los puntos pertenecientes a las dos imágenes.

El mapa será una imagen, nuevamente en escala de grises donde los puntos más claros serán aquellos que estén más cerca y los más oscuros estarán más lejos. Un ejemplo de este tipo de salida lo podemos encontraren la *figura 14*. Esta imagen sacada de la documentación de las OpenCV nos muestra como es este tipo de salida tras su procesamiento en mapas de disapridad.

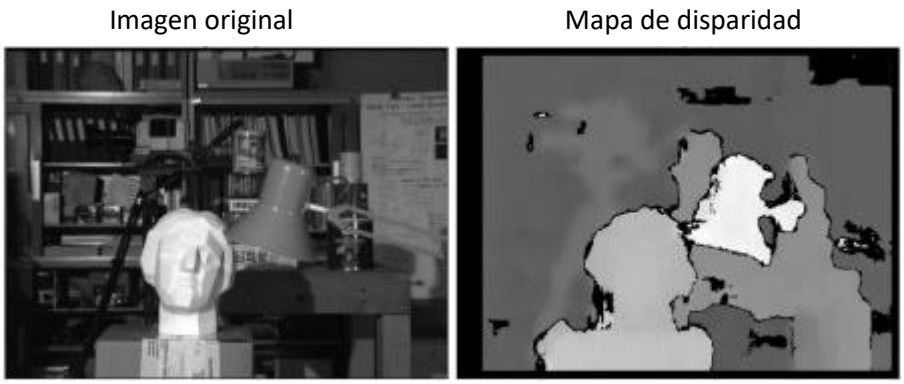

Fig. 4.11 Imagen junto a su mapa de disparidad

La representación que obtenernos a la salida tiene valores de entre 0 y 255, lo que nos produce también imágenes en blanco y negro. Estos valores no quieren decir que nunca se pueda dar un valor de disparidad mayor de 255, pero la representación solo llegara hasta ese valor. No obstante, en las situaciones en las que estamos trabajando, no se darán nunca valores tan elevados.

Estos valores representan la disparidad de cada punto de la imagen, por lo que valores que estén más separados tendrán valores más altos, lo que se traduce en colores más claros. Lo contrario ocurrirá para los no tengan apenas disparidad entre imágenes, que aparecerán más oscuros. [19]

Para implementar este sistema en el programa existen gran variedad de métodos entre los que elegir. Finalmente se optó por uno que el propio equipo de Kitty había realizado benchmarks, o pruebas de rendimiento con él. Se trata de ***Semi-Global Block Matching*** que también forma parte también de las librerías OpenCV.

En estos test se comparan distintos métodos para obtener las distancias. De ellos hemos tenido que filtrar solo los que utilizan imágenes en stereo, y que tengan buenas puntuaciones en la calidad de los resultados. Como criterios adicionales se tuvo en cuenta que no necesitara un hardware muy potente, ya que, aunque la JetsonTX2 está diseñada para estos propósitos, hay gran cantidad de equipos mucho más capaces que también podría usarse en el campo de la investigación que obtendría los resultados mucho más rápidamente pero que tienen un coste monetario muy superior.

Con todo esto en cuenta se eligió el método mencionado que además pertenece a las OpenCV. Así se aseguraría un producto final en el que fuera fácil de integrar todos los módulos que se están diseñando, además que resultaría más sencillo por el trabajo realizado con ellas a estas alturas del proyecto.

Este método consta de dos funciones que son las más importantes. La primera es ***create*** que forma parte de la clase **StereoSGBM**. Esta función es la encargada de preparar al sistema con todos los parámetros de ajuste. Estos parámetros son los datos del sistema físico. Todos estos datos los proporciona Kitty para facilitar su uso para cualquier persona que lo requiera. [20]

Tras crear el sistema con todos estos datos, se llama a la función ***comput***. Esta será la que finalmente cree el mapa de disparidad teniendo en cuenta todos los parámetros fijados anteriormente. Esta función requiere de tres parámetros, todos ellos son del tipo Mat, lo que significa que son el tipo designado por OpenCV para las imágenes. Las dos primeras son las imágenes izquierda y derecha de entrada y la tercera será la salida. Esta tendrá también el mismo tamaño que cualquiera de las otras dos y contendrá tras su ejecución toda la información de disparidad que buscábamos. [21]

Ahora bien, este mapa solo nos aporta información sobre la distancia entre dos puntos en pixeles, nosotros necesitamos magnitudes en metros para poder determinar la distancia a la que se encuentran esos puntos en la realidad. Para ello solo debemos conocer un parámetro más, la distancia focal de la cámara y aplicar unos sencillos cálculos.

Para realizar estos no es necesaria ninguna función compleja, solo un par de transformaciones trigonométricas. Para saber cuáles debemos hacer basta con saber la geometría esquemática del problema y despejar el sistema de ecuaciones resultante. Un esquema del sistema se encuentra representado en la siguiente figura. [22]

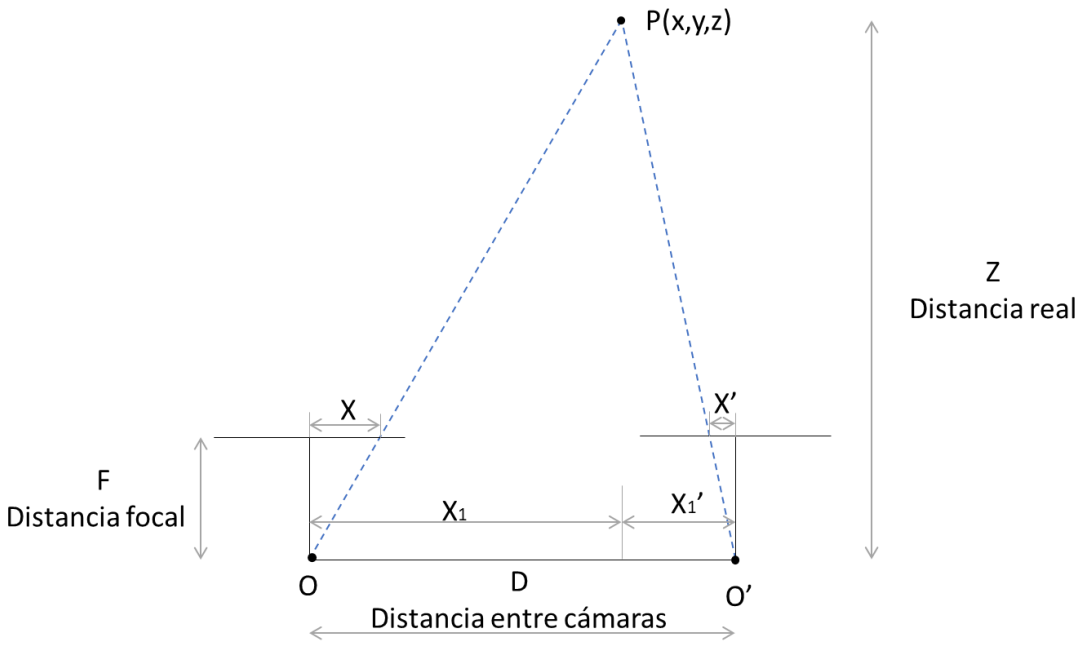

Fig. 4.12 Representación esquemática de los parámetros del cálculo de distancias

Esta es una representación esquemática con una vista cenital del problema. En ella están **O** y **O'** que son las cámaras del vehículo, que están separadas una distancia **D**. El objeto al que queremos determinar sus coordenadas es **P**, que dista $\mathbf{X_1}$ y $\mathbf{X_1'}$ a cada una de las dos cámaras en horizontal. Las distancias **X** y **X'** son la representación de las distancias que han captado cada una de las cámaras desde el centro de cada una de ellas. Por último, **F** es la distancia focal de ambas cámaras.

De este problema lo que nos interesa conocer son las coordenadas del punto P respecto a nuestro vehículo, y los datos que tenemos son F, D y la disparidad que hemos obtenido con el mapa que sería el resultado de restar X y X'. Con esto, vamos a plantear dos ecuaciones utilizando el teorema de Tales y una tercera que relacione las distancias del punto P a las cámaras con la distancia D que existe entre ellas.

$$\frac{F}{X} = \frac{Z}{X_1}$$

$$\frac{F}{X'} = \frac{Z}{X_1{}'}$$

$$X_1 - X_1' = D$$

Una vez planteado podemos despejar las variables.

$$Z = \frac{F}{X} \cdot X_1$$

$$X_1' = \frac{Z \cdot X'}{F}$$

$$X_1 = D + X_1{}'$$

Para después sustituirlas en una única ecuación que nos despeje el valor de Z, la distancia al objeto.

$$Z = \frac{F \cdot D}{X - X'}$$

Debido a que la disparidad $X - X'$ la obtenemos del mapa de disparidad y que en este se obtiene el dato con precisión subpíxel, debemos transformarlo para poder usarlo en nuestras ecuaciones. De acuerdo con la documentación de la función Semi-Global Block Matching debemos dividir el valor obtenido por 16 para obtener resultados en metros, la otra unidad que estamos usando para los cálculos de distancias. Se llamará ***d*** al dato de **salida del mapa de disparidad** que hemos creado

$$Z = \frac{F \cdot D}{\frac{d}{16}}$$

Por tanto, al final tenemos que la distancia al objeto es proporcional a la distancia focal de la cámara y a la separación de ellas, e inversamente proporcional a la disparidad que muestre el punto.

Una vez tenemos la distancia, la coordenada Z, es sencillo calcular la posición relativa del punto respecto del vehículo. Solo hay que volver a aplicar el teorema de tales para tener las coordenadas respecto una de las dos cámaras, por ejemplo, la izquierda. Donde x e y son las coordenadas del punto y X e Y son las coordenadas del punto dentro de la imagen considerando signos desde el centro de la imagen.

$$\frac{x}{z} = \frac{X}{F} \rightarrow x = \frac{X}{F} \cdot Z$$
$$\frac{y}{z} = \frac{Y}{F} \rightarrow y = \frac{Y}{F} \cdot Z$$

Implementando el resultado de estas ecuaciones las coordenadas de cada uno de los puntos vendrán dadas por la siguiente expresión.

$$P\left(\frac{X \cdot F \cdot D \cdot 16}{F \cdot d}, \frac{Y \cdot F \cdot D \cdot 16}{F \cdot d}, \frac{F \cdot D \cdot 16}{d}\right)$$

El resultado que queda es una simple expresión algebraica con multiplicaciones y divisiones, por lo que la carga para el procesador de cálculo será mínima. De esta forma, nuestro programa ya será capaz de asignar unas coordenadas relativas al vehículo a todos los puntos que tenemos disponibles, teniendo así toda la información que necesitamos. Ahora solo necesitamos determinar qué información de toda la que tenemos no resultara útil y cual habría que desecharla mediante un proceso de filtrado de datos.

4.5 Fase IV. Filtrado de la información.

Llegados a este punto, ya se ha extraído toda la información que se requería. Los puntos que se han detectado al inicio con *goodFeaturesToTrack* en las regiones marcadas por la detección de objetos son la principal fuente. De estos datos hemos obtenido sus nuevas posiciones en imágenes sucesivas mediante el uso de *calcOpticalFlowPyrLK*. Por último, de todos los puntos que se tenían en cada momento hemos ejecutado el algoritmo *StereoSGBM* para obtener información adicional de su distancia.

Así finalmente tenemos dos parámetros fundamentales. Por un lado, se tiene su posición en una imagen dada que gracias al mapa de disparidad también nos proporciona su disparidad distancia. Por otro tenemos su desplazamiento entre imágenes, con lo que mirando la diferencia de posiciones podemos obtener los desplazamientos del punto en el tiempo, y por tanto la de los peatones a los que pertenecen.

El problema es que el sistema actualmente tiene puntos que no pertenecen a los peatones, sino que son parte del fondo de la imagen. Además, también comete errores e imprecisiones que habrá que eliminar. Esto puede ocurrir en cada uno de los procesos a los que hemos sometido a las imágenes y es algo normal, pero se tiene que buscar el modo de minimizar sus efectos.

Ahora se van a pasar a detallar las dos principales fuentes de errores que se dan en este programa y su influencia en el mismo.

- El primero de estos fallos es mencionado de puntos que estando dentro de las zonas delimitadas por la segmentación pertenecen al fondo. Esto ocurre debido a que las boundingboxes no se ajustan de forma perfecta al contorno de los peatones, tiene formas geométricas fijas. Por esto hay zonas, sobre todo en las esquinas de cada peatón en las que se estarán estudiando elementos que no son de interés. Estas regiones se han transmitido al sistema para localizar esquinas, por lo que es posible y probable que algunos de estos puntos no pertenezcan al peatón estudiado.

  Durante el proceso de detección de esquinas, aunque se pueden producir errores, estos no van a tener un efecto muy negativo en el resultado. Esto se debe a que, aunque sería optimo que eligiera esquinas muy bien definidas para que el seguimiento en imágenes progresivas fuera más fiable, el punto va estar situado dentro de la zona designada como peatón, por lo que la información que se puede extraer de le seguirá siendo útil igualmente.

  Sin embargo, si afectara algo más al cálculo de los desplazamientos, ya que si no se corresponde exactamente con el mimo punto que en la imagen anterior la medida de desplazamiento no será la real. No obstante, el efecto este último fallo puede tener es mínimo, ya que lo normal es que si se desplaza el punto lo haga por unos pocos pixeles, lo que traducido a distancias en metros no será significativo, aunque dependerá de la distancia a la que se encuentre.

- El otro punto en el que se producen fallos que podemos tratar es durante la creación del mapa de disparidades. Este mapa no es perfecto, ya que el algoritmo de búsqueda de puntos en ambas imágenes puede confundirse y dar distintos errores.

  Uno de los posibles fallos durante esta etapa es que no encuentre correspondencia entre puntos, por lo que la parte del mapa quedaría sin valor, por lo que si tenemos uno de nuestros puntos en esa región no se tendría información de distancia válida.

  El otro fallo que se puede dar durante el cálculo de distancias es que identifique la correspondencia entre dos puntos que no son el mismo. Esto generaría un error al calcular la distancia, pudiendo dar valores mucho más altos o bajos. Esto es aceptable hasta cierto punto, y aunque no vamos a poder detectar si ha cometido un error moderado, si podremos eliminar aquellos puntos que den distancias mucho más altas de lo que es esperado en una determinada región.

Al final todos estos errores se acumularán a la salida del programa, cuando se extraen los datos, ya que por un motivo o por otro un punto puede no ser válido. Por esto, el sistema que se ha implementado en el programa examinará y filtrará los datos una vez han pasado todas las fases de transformación previas.

Para programar este filtrado, se ha subdividido en dos procesos. El primero comprobará que los datos que se han obtenido sean factibles, o tengan valores válidos, mientras que la segunda hará un análisis estadístico para eliminar puntos que se alejen mucho del resto. Este proceso se puede ver representado en el siguiente flujograma.

Fig. 4.13 Flujograma del proceso de filtrado

Cabe destacar que el filtrado que se va a hacer es mediante el análisis en posición de cada punto, ya que es mucho más sencillo de distinguir si un dato es válido o no. Esto se debe a que la distancia a un peatón es un valor que tiene un rango de posibilidades menos amplio, puede estar más o menos cerca, pero siempre estará delante de las cámaras. Así tenemos un rango bastante reducido, lo que facilita el análisis de los datos.

Por el contrario, si analizáramos la velocidad esta puede ser positiva o negativa, ya que un peatón puede estar alejándose o acercándose respecto del vehículo y puede cambiar en distintas partes del peatón, ya que no se trata de un cuerpo totalmente rígido. Puntos situados en brazos y piernas podrían dar valores que difieran del resto del cuerpo del peatón y sin embargo podrían ser correctos.

Además, otro punto a favor de analizar la distancia es que es la base del cálculo de los desplazamientos, ya que estas simplemente se calculan con la diferencia de posiciones de una imagen a la siguiente. Por estos motivos nos resultara más sencillo y fiable realizar el filtrado a los datos originales de posición.

Ahora vamos a pasar a describir el sistema de filtrado que se ha incluido en la programación.

La primera parte del filtrado es bastante sencilla, lo único que vamos a hacer es determinar si los resultados que nos proporciona el sistema para cada punto son factibles o no. Para ello lo que tenemos que hacer es mirar que los valores de del mapa de disparidad y por tanto las medidas de distancias, los valores no deben ser en ningún caso negativos. El algoritmo de ***Semi-Global Block Matching*** nos devolverá el valor -1 si ha ocurrido algún error durante la creación del mapa, lo que nos indicara que ese dato no se ha creado correctamente y por tanto no es válido para su uso.

Otros errores que se pueden tener son valores muy altos, debido a fallos durante la creación del mapa como ya se ha mencionado. Durante las pruebas con este set de datos, han llegado a aparecer puntos a casi 200 metros del coche. Aunque esto pudiera ser factible, en ninguno de los puntos era así, todos se debían a fallos durante algún unto del cálculo de esa distancia. Cabe destacar que estos aparecen de forma muy esporádica, pero que al ser posibles hay que tenerlos en cuenta durante el filtrado.

Los motivos por los que las grandes distancias no son de gran importancia en el proyecto son los siguientes.

- El primero que la localización de las imágenes es entre calles en una ciudad, por lo que las distancias a las que podemos ver a los peatones, son reducidas, es muy fácil que haya obstáculos en medio como mobiliario urbano, árboles o incluso otros coches que imposibiliten grandes distancias de detección.
- Otro motivo es que la detección del sistema de segmentación no lo va a ser capaz de detectar peatones si están tan lejos porque su tamaño en la imagen no lo va a permitir. Nos encontraríamos ante una limitación del software de identificación.
- Y la última y más importante es que en la velocidad están involucradas dos imágenes, con sus respectivos mapas de disparidad. Por esto, hay el doble de posibilidades de que alguno de los puntos tenga alguno de los errores mencionados en al menos una de las dos imágenes. Esto hace que sea más fácil descartar un punto, y de estos no tenemos el doble, sino que son los mismos que se han desplazado en la imagen, por lo que todos los filtros se aplicaran a los datos de distancia de los que disponemos.

Con todo esto, el primer filtro que se va a aplicar lo que va a hacer es reducir todos los puntos a un conjunto con distancias comprendidas entre 0 y 120 metros, un rango que es apropiado para el estudio de los peatones en vías urbanas. Sin embargo, esto sería sencillo de modificar si se requiriera para otra aplicación con uso de mayores distancias, ya que solo habría que modificar el límite superior para el que los puntos se descartan.

El segundo filtro representado en azul más claro en el flujograma es algo más complejo que el primero, ya que tendremos que tratar los datos antes. Para ellos lo que se ha hecho es clasificar todos los puntos que tenemos según a que objeto detectado por la segmentación pertenezcan.

Esto es posible hacerlo, ya que durante la segmentación obtenemos las coordenadas de dos de las esquinas de los extremos que forman el cuadrilátero de cada peatón, por lo que lo único que tenemos que hacer para clasificarlos es identificar para cada punto a cuál de las boundingboxes existentes pertenece.

Esta clasificación se realiza en el momento de la segmentación, ya que en sucesivas imágenes los puntos pueden moverse y salirse de las regiones originales, por lo que en necesario guardar la clasificación de cada punto hasta la próxima vez que se vuelva a realizar la detección.

El proceso de filtrado se realiza de forma individual en cada imagen para cada una de los peatones detectados.

El primer paso del filtro en si es hacer una media simple de todas las distancias, esto lo que nos dará es una primera estimación de la posición del cuerpo necesaria para saber sobre que valores oscilan los datos. Esta será una estimación más o menos fiable debido a que no van a aparecer valores muy elevados, ya que ya los hemos retirado en el primer filtro, y aunque coja distancias del fondo de la imagen, al tener una gran cantidad de puntos se suavizara su efecto en la media.

Tras esto, se calculará la desviación típica o desviación estándar de cada objeto estudiado, para lo que es necesaria la media que acabamos de calcular. Esto nos servirá para poder tener un valor que dependa de cada muestra y que podamos usar como referencia para determinar que valores están muy desviados respecto a la media de cada muestra para poder eliminarlos.

Lo que haremos será calcular el sumatorio de todas las diferencias al cuadrado de todos los datos con la media obtenida anteriormente y el resultado lo dividiremos entre el número de puntos que tengamos y le haremos la raíz cuadrada.

$$Desviacion\ estándar = \sqrt{\frac{\sum_{i=0}^{n}(X_i - \bar{X})^2}{n}}$$

Con esto obtendremos una medida de lo dispersos que están los datos que tenemos. Ahora este valor lo podremos utilizar para eliminar aquellos valores que se considere que se desvían demasiado como para pertenecer al peatón. Así tenemos un sistema flexible que se adapta a las características de los datos de cada grupo. Este sistema requerirá más precisión a los conjuntos de datos más compactos, pero que sea más permisivo con aquellos en los que los datos están más dispersos.

Para eliminar los datos que más se alejan no se va a utilizar directamente la desviación estándar, sino que se va a multiplicar por un factor. Esto permitirá modificar aún más la sensibilidad que queremos tener en el sistema de filtrado haciéndolo más versátil en futuras aplicaciones.

Tras probar el algoritmo en esta serie de imágenes con diversos valores, se observó que con valores superiores a 1.2 eliminaba demasiados puntos, y que con 1.1 aún quedaban bastantes fuera de los contornos de los viandantes. Por ello en el programa se ha establecido la constante que multiplica a la desviación estándar como 1,15.

Además, para intentar obtener resultados mejores, este proceso de filtrado ejecuta dos pasadas correctoras, con lo que la media se recalculará una vez se han eliminado los primeros valores para hacer otra segunda pasada afinando aún más. Solo se realizan dos pasadas, ya que a partir de ahí los resultados que se obtenían no mejoraban significativamente, ya que se empezaban a eliminar casi por igual puntos validos como puntos del fondo de la imagen.

Para visualizar mejor este filtrado, se muestra la siguiente imagen, la *figura 17*, que nuevamente se trata de la primera imagen del set. En ella se pueden ver marcados todos los puntos que han sido detectados, pero utilizando dos colores distintos. En verde todos aquellos que se han considerado como válidos, y que han pasado todos los filtros, mientras que en rojo se muestran aquellos que no van a ser considerados como salida valida del programa y que por tanto son eliminados.

Fig. 4.14 Primera imagen tras realizar el filtrado de puntos

En ella están representados un total de 183 puntos, de los cuales 142 se han considerado válidos y marcados en verde, y 41 han sido marcados en rojo y por tanto descartados. Se puede ver como la mayoría de estos puntos, están situados sobre el grupo de personas de la derecha.

A continuación, hay un detalle de esa zona de la imagen para poder analizarla mejor.

Fig. 4.15 Detalle del filtrado de la primera imagen

Se puede apreciar como la mayoría de los puntos rojos recaen sobre el fondo. Entre los puntos eliminados destaca una traza casi vertical de puntos que han caído sobre un peatón que está más al fondo de la imagen. Esto es perfectamente válido, ya que este no había sido detectado por el proceso de segmentación por estar prácticamente tapado por los peatones del frente, y dado que está más atrás, la distancia de estos puntos no corresponde con la de los demás puntos de la boundingbox a la que pertenecen.

En la parte central se eliminan con éxito puntos sobre el segundo peatón desde la izquierda y entre su brazo y el cuerpo.

En la parte de más a la derecha, los puntos están más dispersos, y en su mayoría apuntan a zonas que están en el borde de los peatones. Esto lo que nos hace ver es que, aunque estén sobre el contorno del peatón, la medida de distancia que hace el mapa de disparidad, habrá tomado la distancia a la pared trasera y no la del peatón. Esto se puede corroborar con los datos numéricos, que muestran que las distancias de los puntos descartados son significativamente superiores a la media del conjunto del peatón.

Solo tenemos un punto que se encuentra directamente sobre el cuerpo del cuarto peatón, pues, aunque nos elimina muchos puntos que no son válidos puede tener algún falso negativo que se encontrará cerca de los márgenes establecidos. Esto sin embargo no resultara un gran problema por la gran cantidad de puntos que aún quedan clasificados como válidos.

También hay que destacar que no se han eliminado todos los puntos ajenos a los peatones. Estos falsos positivos se pueden deber fundamentalmente a dos motivos.

El primero que el fondo se encuentre muy cercano al peatón, este es por ejemplo el caso de los puntos del pavimento sobre el que andan, que, aunque no es el peatón en distancia se encuentran muy cercanos a él.

El otro son los fallos en la generación de las distancias, ya que se trata de un método de detección óptico, y la fiabilidad de este no es tan alta como utilizar un sensor especializado en este tipo de mediciones. Sin embargo, con las cámaras podemos hacer estas mediciones además de todas las otras identificaciones, por lo que seguirá siendo válido para el proyecto.

Al final, con este procedimiento de filtrado habremos conseguido reducir el número de puntos que se habían detectado con el algoritmo de Harris de detección de esquinas y que estaban dentro de la zona clasificada como peatón, pero que realmente forman parte del fondo, mejorando así la precisión del programa sin necesidad de modificar de forma física ninguna parte del sensor, solo empleando el software.

Un análisis mas detallado de los resultados de este proceso se puede leer en el capítulo siguiente, el *capítulo 5, análisis de resultados* donde se estudiarán a fondo las salidas del programa.

4.6 Salida de datos y cálculo de velocidades.

Tras finalizar el módulo anterior, ya se han realizado todos los tratamientos pertinentes de los datos que se han obtenido. Lo único que queda por hacer es programar la forma en la que saldrán del programa para que puedan ser utilizados.

Esta salida del sistema debe presentar los dos datos que hemos perseguido durante la creación de este proyecto, la posición y velocidad de los peatones.

La medida de distancia es trivial, ya que es el último dato que hemos obtenido tras el proceso de filtrado, por lo que no requiere de ninguna transformación adicional. Por el contrario, el dato de desplazamiento, si requerirá unas simples transformaciones para poder obtener la medida de lo que se ha desplazado cada peatón entre una imagen y la anterior.

Para obtenerlo, no se van a utilizar los datos medios de posición de cada peatón, si no que emplearemos los puntos individuales de los que se compone esta medida.

Por la forma en la que se ha programado, podemos conocer la asociación de cada punto al peatón identificado correspondiente. Además, al no haber eliminado ninguno de los puntos detectados, aunque se hayan descartado, solo tendremos que calcular la diferencia en posición de ambos puntos para calcular su desplazamiento asociado.

Para esto solo hay que comprobar que se cumpla una condición, que ambos puntos, el de la imagen actual y el de la anterior hayan sido clasificados como válidos. Tras esto se podrán restar los valores y se repetirá con cada punto de cada peatón. Al final, se volverá a calcular la media de todos los desplazamientos y se obtendría el resultado de medio correspondiente a cada peatón de la imagen.

Aunque de esta forma se ha obtenido el dato buscado de desplazamiento, y durante la memoria se ha utilizado como sinónimo de velocidad de los cuerpos, esto no es del todo preciso. Para obtener el dato de velocidad solo tenemos que dividir el movimiento de cada peatón, expresado en metros, por el tiempo entre imágenes, para obtener una medida adecuada para la velocidad. Sin embargo, el proyecto de Kitty no proporciona ni el tiempo entre imágenes ni la tasa de fotogramas por segundo a las que han sido tomadas las imágenes. Por este motivo solo se ha podido llegar a expresar los resultados como medidas de distancia en metros. No obstante, si se conociera este dato sería tan sencillo como dividir todos los resultados por la medida de tiempo para conseguirla, un proceso trivial para los ordenadores.

Además de estos datos, de forma indirecta también se está presentando otro que podría ser interesante y empleado en otras aplicaciones, el número de grupos de peatones. Este dato, aunque no nos determina el número exacto de viandantes que aparecen en la imagen, si nos puede dar una idea de la cantidad de grupos de personas presentes en cada instante.

Finalmente queda por mencionar la forma en la que estos datos son presentados en el propio código. Debido a la naturaleza del programa, diseñado para su implementación en otras aplicaciones, se podría hacer uso de los datos de salida directamente de los vectores de valores en los que se almacenan durante la ejecución. Sin embargo como se trata de la salida actual del programa, también se ha procurado su salida en documentos de texto para poder analizar los datos de estas salidas tras la ejecución del programa.

4.7 Estructura del software. Gestión de los datos y el flujo del programa.

Ya se han descrito todos los módulos para obtener la solución final. Sin embargo, su conexión en un único programa no es tan sencilla como pudiera parecer. Para interconectar los distintos módulos no basta con escribir un código tras otro de todos los módulos para obtener un programa que tenga todas las funcionalidades que hemos creado de forma individual.

Como ya se comentó anteriormente, la base del programa es el sistema de detección de peatones de Jetson Inference, y dado que es la única porción de código que vamos a reutilizar en el proyecto, ha sido una buena base para comenzar. No obstante, para que se adapte al flujo del programa, no basta solo con hacer las modificaciones a las a salidas que obtenemos de él.

El programa de segmentación está diseñado para solo realizar una única ejecución, pero nosotros necesitaremos que el programa sea cíclico y se repita para la serie de imágenes que en este caso estamos usando. Por este motivo tampoco podemos poner en primer lugar la segmentación, necesitaremos algún tipo de bucle que nos permita reutilizarlo cuantas veces sea necesario.

Para hacerlo, y de forma eficiente, se ha separado el núcleo de este software en dos partes.

- La primera es un proceso de carga de toda la red de identificación de objetos, todas las librerías y subprocesos que son necesarios tener en memoria para poder ejecutar la detección propiamente dicha. Esta parte solo es necesaria hacerla una vez, y formaría parte de la preparación inicial del programa. De este modo estamos ahorrando una gran cantidad de tiempo a cada uno de los sucesivos ciclos que ya dispondrán de toda la información sin tener que volver a leer del disco duro.

- La segunda es el propio proceso de identificación y procesado de las imágenes y los datos extraídos. Esta es la fase que realmente nos interesa, ya que es la que nos aportará la información. Sin embargo, este proceso a su vez se puede volver a subdividir, ya que en un ciclo de ejecución podría ejecutarse una reinicializarían de los peatones en la imagen o reconstruirlos de la imagen anterior. Por esto habrá que determinar cuándo es necesario realizar este proceso de inicialización de datos y cuando no para ahorra en tiempo y recursos de cálculo.

Este proceso de inicialización no es necesario hacerlo cada vez que se ejecuta el algoritmo, solo nos sirve para recolocar los puntos en nuevas regiones, pero cuando no lo ejecutamos podemos seguirlos con el método del flujo óptico. Por tanto, reducir el número de veces que hagamos este proceso agilizara mucho el tiempo de ejecución manteniendo todas las funcionalidades del programa.

La estructura interna del programa quedara así condicionada a cuantos ciclos queramos hacer entre sucesivas inicializaciones. En este caso se ha elegido que ese número sea 3, ya que es un numero lo suficientemente pequeño como para evitar problema de pérdidas de puntos porque los peatones salgan de la escena y que se pierdan otros que entren nuevos en otra parte de la imagen, a la vez que se mantiene relativamente bajo para no ralentizar innecesariamente el programa con cálculos más complejos.

De esta forma tendremos una estructura básica del programa que se puede representar mediante el siguiente diagrama de flujo.

Fig. 4.16 Diagrama de flujo del programa

En él podemos ver representado en verde el proceso de carga de la información fundamental del sistema de segmentación que solo se realiza una vez. Tras esto se llega a la estructura cíclica, en la que existen las bifurcaciones para determinar si se buscan nuevos peatones o no. Tras esto se realizarán el resto de procesos de la ejecución, cálculo de distancias, velocidades y filtrado de datos antes de proceder a su salida del sistema para ser utilizados. Tras completarlo, el programa volverá a ejecutarse hasta que se termine su ejecución, o no queden más imágenes que leer.

Esta es la estructura básica del programa. Un código cíclico que regula toda la información para procesarla y presentarla al final de cada iteración. Con todo, siguen quedando dos dificultades que se plantean para conseguir un resultado único y cohesionado que se expondrán en los apartados siguientes, la inicialización de las imágenes y la organización de los datos que se comparten entre los diversos módulos que componen el programa.

4.7.1 Inicialización de las imágenes

Al inicializar el proceso de segmentación, se vuelven a calcular todos los puntos, desechando los que teníamos muestreados en las 3 imágenes anteriores. Esto por sí solo no representa un problema, ya que desechamos unos datos, pero tenemos otros nuevos para suplirlos, lo que nos permite tener valores de la posición de todos los peatones detectados. Sin embargo, para calcular los desplazamientos se necesitan los datos de dos imágenes, los de la imagen actual y los de la anterior para poder calcular la diferencia de posiciones.

Si se ejecutara el algoritmo de forma secuencial, nos dejaría con datos de posición en todas y cada una de las imágenes, pero no de desplazamiento. Debido a que no hay ninguna correspondencia entre los puntos nuevos y los antiguos, no se podría calcular la velocidad en 1 de cada 3 imágenes que se analizaran, lo que se corresponde con los ciclos de inicialización.

Por esto, hacerlo de forma secuencial no tendría sentido, ya que estaríamos perdiendo un tercio de toda la información posible relativa a la velocidad. Se debe implementar una solución por tanto que permita saber el valor de la velocidad de cada peatón identificado en todas las imágenes.

Para lograr esto, lo que se ha hecho es calcular la nueva posición de los puntos antiguos como si se tratase de una imagen más justo antes de iniciar el proceso de búsqueda de nuevos más peatones. De este modo en la primera ejecución de cada conjunto de tres imágenes se tendrán dos simultáneamente dos conjuntos de puntos de la misma imagen.

Los que se han obtenido previamente a la inicialización se usaran en conjunto con las de la imagen anterior para calcular los desplazamientos. En cambio, las que se han obtenido tras la segmentación, se usaran para la distancia de la nueva imagen y se almacenara para su posterior uso en el cálculo de la velocidad de la imagen siguiente.

En el siguiente diagrama se ha representado el flujo de datos de la solución adoptada que se ha descrito y se detallara a continuación.

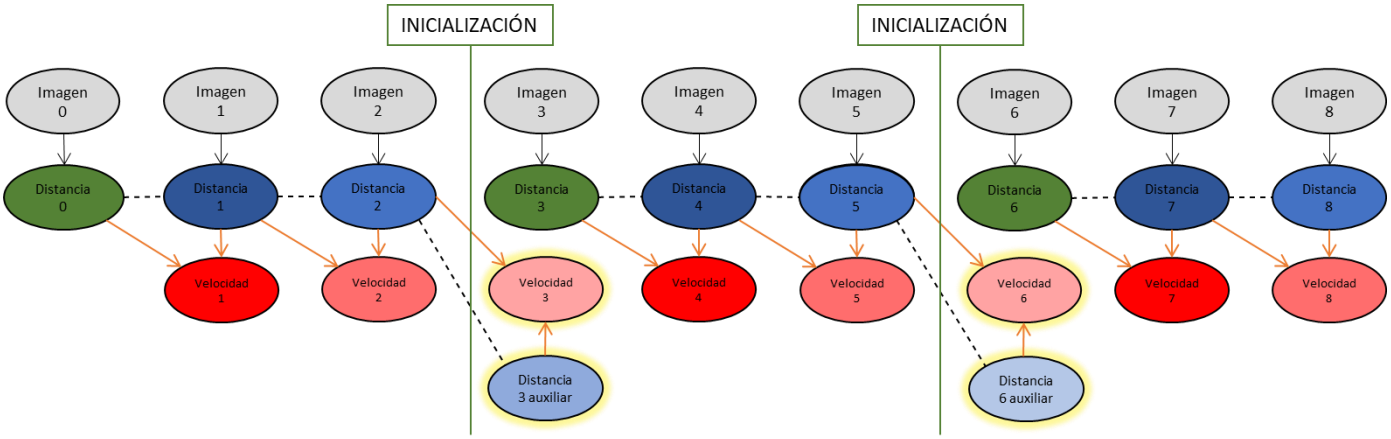

Fig. 4.17 Flujo de datos en los distintos ciclos del programa

En el diagrama se han representado los conjuntos de datos que representan la información, así como también las relaciones entre los mismos. Su organización por columnas agrupa todos los que datos que pertenecen a un mismo ciclo de ejecución del programa y por tanto a una misma imagen.

En primera fila se han representado únicamente las imágenes que componen la secuencia de las que proviene toda la información. De estas imágenes se ha extraído la posición y distancia de los puntos. Estas distancias están representadas en la segunda fila en tonalidades de azul, a excepción de la primera que, en color verde, representa la inicialización de los nuevos puntos obtenidos mediante la segmentación y la posterior detección de esquinas.

Las distancias están relacionadas con las velocidades que se han situado en la tercera fila. Se puede a preciar como de la primera imagen de cada grupo de 3 aparece un conjunto adicional de datos resaltados en amarillo. Estos son la velocidad de esa primera imagen y las coordenadas actualizadas de posición que la hacen posible, situadas de arriba abajo en el diagrama.

Cabe destacar que esta mejora no es posible aplicarla a la primera imagen que se procese en un conjunto de imágenes, puesto que no existen datos previos que podamos usar para calcular los desplazamientos de los puntos. Así que tendremos un tiempo de puesta en marcha del sistema de un ciclo adicional desde el cual ya podremos obtener todos los datos requeridos en todas las imágenes sucesivas.

4.7.2 Organización de los datos

La otra gran dificultad de la unificación de todos los módulos es el tratamiento de todos los datos que se manejan. Tanto la salida de la función goodFeaturesToTrack, como la de calcOpticalFlowPyrLK nos devuelven un único vector con todos los puntos que ha obtenido. Sin embargo, estos datos no tienen ningún orden concreto, salen en el orden en el que el algoritmo los ha detectado, pero esto no quiere decir que los que estén más cercanos en la imagen vayan a estarlo en el vector de salida.

Esto hace necesario volver a copiar todos los datos en otro vector diferente para tenerlos en un orden que nos resulte más lógico y conveniente para las operaciones, mientras se conservan los datos originales por si fueran necesarios. En el programa esto se ha hecho asignando los puntos de cada imagen a un vector con dos dimensiones. La primera es la que contiene a cada uno de los peatones, y a su vez cada uno de estos contiene todos los puntos que se encuentren dentro en la imagen. Además, en el programa necesitamos hacer uso de los datos de dos imágenes de forma simultánea para el cálculo de los desplazamientos, tres en el caso de que se acabe de realizar una inicialización como se ha explicado en el apartado anterior, lo que nos obliga a convertir a este vector en uno tridimensional.

Esta organización se puede ver representada en la siguiente figura.

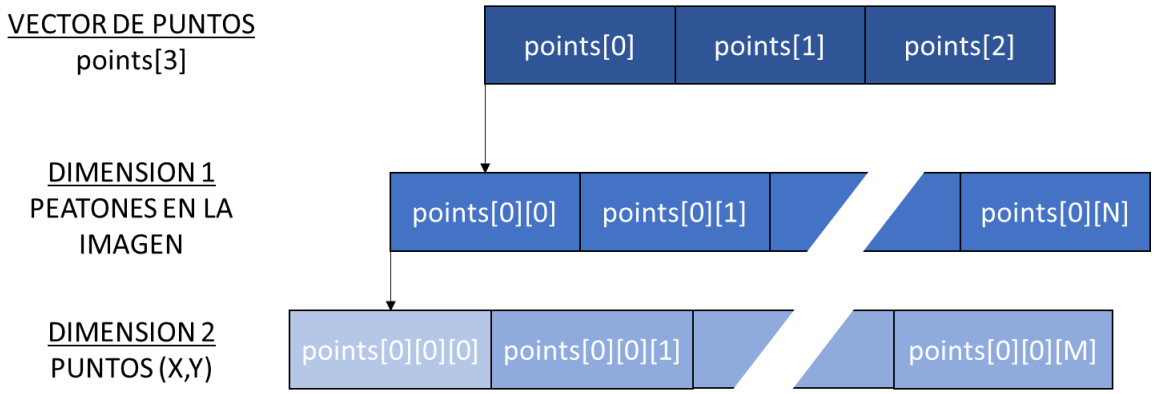

Fig. 4.18 Representación de la estructura de datos de los puntos de interés en el programa

Por lo tanto, en todo momento se tienen como mínimo los datos de dos imágenes en las que hay que tener controlado su orden, y asociación a cada peatón de cada uno de ellos. Esto no sería tan problemático si los puntos no pudieran perderse de una imagen a otra, pero como se ha comentado en apartados anteriores, es algo que puede ocurrir debido a la gran cantidad de ellos que se procesan.

Para solucionar este problema, hay que revisar todos y cada uno de los puntos en el vector de *status* a la salida de calcOpticalFlowPyrLK y comprobar que este se haya encontrado al trazar el flujo óptico. De no encontrar el punto, se ha codificado que se modifiquen sus coordenadas x e y en la imagen por un -1 en ambos. Al estar fuera del rango de posibles valores que pueden asignarse de forma natural, nos aseguramos de poder distinguir el punto de otros que si tengan sus asociaciones correctamente identificadas.

Un punto importante que hay que destacar es que el punto no se elimina. Si se hiciera haría que todos los puntos sucesivos se emparejaran con otros que no tienen ninguna relación, causando que las lecturas de desplazamientos no fueran coherentes.

Otro punto con gran carga en la cantidad de datos es el proceso de filtrado. Durante su ejecución se deben hacer copias nuevamente de los datos, en este caso es totalmente necesario, ya que se pretende eliminar puntos que son válidos, puntos que han pasado todas las restricciones anteriores pero que no entran dentro de los límites de los filtros establecidos.

Además, se debe mantener el registro de cuantos datos han pasado los filtros para saber cuántos puntos componen tanto la media como la desviación típica para poder calcularlas y almacenar sus datos. Esto hay que hacerlo tres veces por peatón que aparezca en cada imagen, además de tener que realizarlo otra vez desde el inicio en el caso de que se trate del ciclo de inicialización donde habría que tratar los puntos auxiliares adicionales.

La última gran dificultad es coordinar los procesos que hay que hacer en cada ciclo. Este problema viene dado porque no en todas las ejecuciones del programa hay que realizar todas las tareas, ya que el ciclo de inicialización es distinto a los demás. Esto sería más sencillo si se considerara que el número de imagen entre inicializaciones fuera fijo. Esto sin embargo no se ha hecho así, sino que se ha hecho en función de parámetros, lo que dotara al sistema de una mayor flexibilidad en futuras modificaciones.

En realidad, este no es un problema técnico que requiera de más funciones, o mayor carga de computo. Simplemente se trata de una cuestión de manejar más dimensiones aun, lo que hace que en conjunto con todas las que tienen los vectores hace que sea más sencillo equivocarse y además sea más difícil encontrar las causas del fallo.

Estas serían las consideraciones más importantes que se han tenido en cuenta durante el diseño del programa para dar solución al objetivo de monitorizar el entorno en busca de peatones y poder tener una información instantánea de su posición actual y su desplazamiento.

En el *capítulo siguiente* se analizarán los resultados que se han obtenido, sacando conclusiones sobre ellos y las decisiones que se han tomado durante el diseño del conjunto del programa.

CAPÍTULO 5: Análisis de resultados

En este capítulo del trabajo se analizarán los todos los datos obtenidos mediante el software creado, estudiando la fiabilidad de los mismos. Este análisis va a tener que ser en su mayoría un estudio manual de los datos de salida dado que la base de datos de Kitty no dispone de las medidas reales de distancias para esta secuencia de imágenes en lo relativo a los peatones, por lo que no podremos crear un código para que compruebe el acierto de los datos de forma automática.

En su lugar se tendrá que recurrir a analizar las secuencias de salida de forma visual, comprobando la veracidad de las salidas, por lo que es más fácil que se produzcan errores durante la toma de estos datos, debido al factor humano.

El programa se analizará de forma modular, de igual forma a como fue creado. Esto nos permitirá determinar la calidad de las transformaciones que se obtienen con cada uno y ya en el apartado de conclusiones se valorará el programa en su conjunto.

El primer apartado del que vamos a comprobar la precisión de los datos es del apartado de detección de peatones.

5.1 Sistema de detección

La detección de peatones es la base fundamental del proyecto, ya que nos permite sacar los datos básicos para poder aplicar el resto de los sistemas de procesamiento de la información.

Además, realizar este proceso para un ordenador no resulta tan trivial. Nosotros podemos reconocer personas que estén parcialmente cubiertas, en posturas poco comunes o en ambientes con condiciones de luz complejas. Todo esto son puntos que hacen que los sistemas de detección de objetos o segmentación sufran para encontrar los objetivos de su búsqueda en las imágenes.

En este conjunto de capturas por ejemplo tenemos dos de estos problemas en prácticamente todas las imágenes. Por un lado, las condiciones de luz son bastante complejas. Al tratarse de una zona arbolada en un día muy soleado, se crean en una misma imagen zonas oscuras en las que no incide el sol quedando a la sombra, pero en otras zonas de la imagen si da directamente el sol saturando la zona de la imagen.

El otro problema es que se interpongan objetos delante del propio peatón. Esto es algo que es habitual en entornos de ciudad, ya que objetos del mobiliario urbano como farolas o arboles suelen encontrarse en las aceras, además de vehículos aparcados, lo que en algún momento supondrá que se pierda la línea de visión entre la cámara y los peatones.

Un ejemplo de ambos problemas lo podemos encontrar en la imagen 21 del set, donde se produce la inicialización de Segnet en busaca de peatones.

Fig. 5.1 Imagen 21 de la base de datos con detección de peatones

Podemos ver como las dos personas de la derecha no se detectan, ya que aparecen completamente blancos y se confunden con la acera que se encuentra también al sol y de un color blanco intenso.

De estos, el que se encuentra más cerca si se había reconocido en imágenes anteriores, ya que no aparecía tan sobre expuesto. Por el contrario, el peatón más alejado camina en todas las imágenes por el sol y siempre aparece completamente blanco, confundiéndose con el pavimento y los edificios todos saturados en color blanco, lo que hace que no se diferencie de ellos.

El problema de ocultamiento parcial de peatones también se encuentra en esta imagen, ya que el par de peatones de la izquierda se tapa con un árbol, y no consigue detectar a ninguno de los dos cuando si lo había hecho en imágenes anteriores. Lo mismo ocurre con los tres peatones que hay más adelante en la acera derecha, pero con la diferencia de que en este caso si consigue reconocer a al menos uno de ellos.

Una vez hemos identificado los problemas a los que nos enfrentamos, podemos pasar a revisar las imágenes. Estas no serán las 180, ya que se inicializa tan solo cada 3 imágenes, por lo que, aunque se trate de un tercio del total se tendrán que seguir revisar de forma manual 60 imágenes. En ellas se buscará ver cuántos peatones han quedado sin identificar, y cuantas de las identificaciones han sido falsos positivos, detecciones que no eran peatones.

Después de analizar estas 60 imágenes se han obtenido los siguientes resultados recogidos en la tabla.

	Grupos detectados	porcentaje
Total detectados	124	100
Falsos positivos	6	4.8
Falsos negativos	35	28.2

Tabla 3 Resumen de peatones detectados en el set de imágenes

Tenemos que en total se han detectado 124 grupos de peatones. Esto no quiere decir que en total contando todas las imágenes haya 124 peatones, ya que estas detecciones en ocasiones engloban a más de una persona. De esta forma, el número de peatones detectados seria mucho mayor a este, pero a fin de poder hacer una comparativa en igualdad de condiciones, se va a realizar con grupos en lugar de personas.

En primer lugar, vamos a ver los falsos positivos, estos son todos los grupos que han aparecido en una o más imagen que no contenían peatones, pero que estaban identificados como tal.

En total se han detectado 6 de estos grupos, lo que representa menos del 5% del total. De estos la mitad han sido debidos a identificaciones de bicicletas estacionadas como vehículos que estaban siendo utilizados por peatones.

Este es el caso de la imagen 84 que se muestra en la *figura 23*. En ella se identifica a dos grupos de peatones, un ciclista y también a la bicicleta parada que no está siendo utilizada. Este es un caso de falso positivo, porque realmente no forma parte del grupo de objetos que queremos monitorizar. Sin embargo, no será un elemento que induzca a error, ya que aunque no se mueva, es un elemento que tampoco querremos que se interponga en la trayectoria del vehículo, por lo que su monitorización el único efecto negativo que tendrá es que se están consumiendo recursos en su control.

No obstante, esto no justifica eliminar del reconocimiento a los ciclistas, ya que sí que resultara útil poder detectar a aquellos que si circulen por aceras o por calzadas.

Fig. 5.2 Falso positivo en la imagen 84

Los otros 3 errores restantes son debidos a que delante del vehículo con las cámaras está circulando un autobús con publicidad en su parte trasera. En ella aparece una persona, por lo que, aunque técnicamente está haciendo bien su trabajo detectando personas, no se trata de un viandante. Este error ocurre en tres imágenes, entre ellas la imagen 159 que se muestra a continuación.

Fig. 5.3 Falso positivo en imagen del autobús, imagen 159

Este error aunque tampoco es crítico, se podría solucionar si también se identificaran vehículos, ya que se podría ver que siempre tiene las mismas coordenadas que uno de ellos por lo que se podría descartar. De cualquier forma, es un caso que, aunque como se ve, es posible, es mucho más difícil que se dé en circunstancias normales.

El último punto a analizar en este apartado son los falsos negativos, es decir, todos aquellos peatones que no han sido detectados, pero que se encontraban en la imagen. Para ello, hay que tener en cuenta las limitaciones de las que hemos hablado anteriormente.

Para hacer un análisis lo más justo posible, se han contabilizado las veces que un peatón o grupo de peatones no es detectado. Esto se ha hecho contado para todas las imágenes posteriores a una en la que se había detectado dicho grupo. En caso de que no se llegue a detectar en ningún punto se ha contabilizado en todas las imágenes en las que se inicializa en las que aparece sin ningún obstáculo delante.

Así el número de grupos que no se han detectado sale 35, lo que supone que se podrían haber detectado un 28% más de grupos de peatones de los que se ha hecho. Este es un dato bastante alto, y no puede ser ignorado, pero viene agraviado en gran medida por la secuencia de imágenes que comprende desde la 72 a la 108. En ellas aparece un grupo de peatones que no se detecta en ningún momento.

Este grupo muy seguramente no es detectado porque se juntan los dos problemas ya mencionados. En la zona de la imagen en la que aparecen está expuesta directamente al sol, y además su fondo es una zona con sombra, además llevan ropa oscura que hace que sea muy difícil de identificar en esas circunstancias. Esto hace que sea difícil distinguir formas en ese grupo de personas. Además, los peatones que conforman el grupo andan muy juntos unos de otros, por lo que nunca se llega a ver un peatón de forma individual, lo que puede estar produciendo estas no identificaciones.

La imagen 96 se muestra a continuación en la *figura 25* y sirve para visualiza mejor el error al que se está haciendo referencia.

Fig. 5.4 Identificación fallida a grupo de peatones en imagen 96

Si no tuviéramos en cuentan tan solo ese grupo de personas, el porcentaje de personas sin detectar bajaría hasta un 20%, lo que lo que sería una proporción mucho más aceptable a día de hoy en este tipo de sistemas inteligentes. Cabe destacar que hay tan solo 7 peatones en todas las imágenes en las que no se apreciaran condiciones muy adversas en las que el sistema no ha sido capaz de detectarlos. Este dato se acercaría al 5% de error de los falsos positivos.

Finalmente, hay que tener en cuenta que este sistema no es diseño propio, si no que la parte de código de identificación de la Jetson inference y que no estaba diseñado exclusivamente para un sistema de detección de peatones, por lo que existe un margen de mejora importante.

Sin embargo, era importante analizarlo, para poder poner los demás datos en contexto. Sin duda se podrían podría mejorar de distintas maneras. Una de ellas seria entrenar al sistema de reconocimiento en situaciones lumínicas adversas, lo que haría aumentar las tasas de acierto en estos entornos.

Otra mejora que se podría aplicar sería el transformar el sistema de detección en un sistema continuo. Este tendría en cuenta no solo la información de la imagen actual, sino también de otras imágenes anteriores. Se podría tener un sistema que disminuyera los requisitos de seguridad a la hora de identificar un posible peatón si anteriormente se ha detectado uno en una determinada zona y en siguientes imágenes no aparece ninguno con el coeficiente de seguridad suficiente. Esto habría que acompañarlo de sistemas adicionales para evitar tener más falsos positivos y que solo permitieran esta circunstancia si se sabe con relativa seguridad que esa supuesta región del espacio corresponde con la misma zona del entorno, esto es que el vehículo no ha cambiado radicalmente de orientación.

5.2 Sistema de posicionamiento y seguimiento de puntos

El sistema empleado para la distribución de los puntos por las imágenes tiene poco que se pueda destacar. Es un sistema que, aunque necesitamos que las esquinas que elija sean lo más distinguibles posibles no necesitamos información de un punto en concreto, sino de toda la región. De este modo lo único que nos interesa es que los distribuya de una forma uniforme y con grandes porcentajes de calidad en los puntos que el sistema elija, para evitar su perdida.

Por su parte, el sistema de seguimiento de los puntos una vez ya inicializados, tiene bastante fiabilidad durante su uso. Durante la ejecución de la secuencia, rara vez se ha apreciado que algún punto que estaba en ambas imágenes se haya colocado en una región muy distinta de la imagen. Esto ocurre en contadas ocasiones, pero suele darse en circunstancias en las que cambia la iluminación del peatón, o el fondo de la imagen en la que está situado. Aun cuando estas condiciones se dan, y el sistema no lo acierta de forma perfecta, siempre suele colocar los puntos en el peatón, aunque sean en una región cercana a la original. Esto es gracias a que normalmente sí que se existe una diferencia clara entre las texturas que tienen el peatón y su entorno.

No obstante, existen situaciones en las que el seguimiento de puntos no tiene tan buenos resultados. Estas situaciones son cuando los pixeles que se estaban siguiendo no se encuentran en la imagen siguiente. Esto era de esperar, ya que no puede detectar unos puntos en la imagen si no se encuentran en ella. Esto ocurre principalmente porque se interponen objetos ajenos a los peatones seguidos, por ejemplo, árboles o farolas. Cuando esto ocurre, los puntos que estaban en la región del peatón y que en la siguiente están ocultas por cuerpo que interrumpe la visión, se redistribuyen por la zona cercana que le corresponde a cada uno.

Esta situación en la que se interpone un objeto ajeno delante de un peatón se ve representada en la siguiente figura.

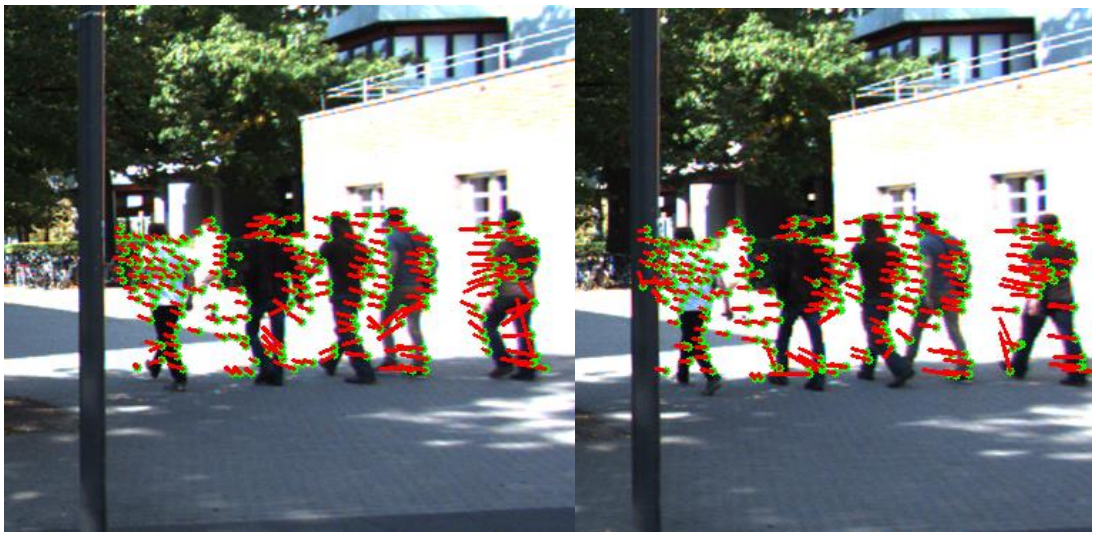

Fig. 5.5 Detalle del flujo óptico entre las imágenes 4 y 5

Esta figura está formada con detalles de la zona derecha de las imágenes 4 y 5 del set de datos. En ellas se pueden apreciar los dos tipos de errores que se han descrito. En primer lugar, a la izquierda de cada una se puede ver como los puntos que estaban cercanos al peatón en la imagen 4 en la siguiente aparecen sobre la farola, ya que los pixeles del peatón a los que pertenecían ya no se pueden ver y por tanto se colocan encima del objeto extraño que es la zona más próxima a su anterior ubicación.

Por otro lado, en el peatón situado más a la derecha en cada imagen se ve como algunos de los puntos que estaban situados sobre la zona baja de la pierna se terminan desplazando a hacia el pie.

Estos problemas son difíciles de evitar. El de puntos que no se corresponden es un fallo del algoritmo al encontrar el lugar al que se han movido. Si bien es verdad que se podría cambiar parámetros para intentar reducirlo o diseñar otro algoritmo que obtuviera mejores resultados, las veces que esto ocurre son muy bajas. Esto junto con que se tiene una densidad de puntos bastante alta por imagen hace que no influya apenas en el resultado del programa puesto que si sigue en el peatón seguirá siendo información útil para el cálculo de su distancia, y si por el contrario ahora forma parte del fondo, el punto se filtrará y no se tendrá en cuenta.

5.3 Sistema de cálculo de distancias

Para analizar la calidad de los resultados de distancia, no disponemos de los datos reales de las distancias a todos los puntos. Sin embargo, si tenemos benchmarks que han realizado el equipo de Kitty database para comprobar la fiabilidad de los datos que se obtienen con el método empleado en un entorno de conducción autónoma.

Según sus datos, realizados con un procesador de un solo núcleo a una frecuencia de 2.5 GHz, tiene un porcentaje de fallo de media de 10.86%. Esto supone que poco más de la décima parte de las distancias que se calculan, el dato de disparidad obtenido esta 3 o más pixeles desviados respecto al valor real. Según el estudio el criterio empleado ha sido considerar como dato erróneo cuando la distancia medida sea superior al 5% a la real en la imagen.

Los datos de estos test los tienen más desglosados e indican que los objetos al frente de la imagen ronda más el 20% mientras que el fondo de la imagen solo tiene un 9%. Esto es algo positivo del sistema ya que, aunque se comenten más errores en objetos cercanos, un error en este entorno resultara en una menor desviación en metros que si ocurriera en un objeto lejano. Además, generalmente se tienen una mayor cantidad de puntos en los cuerpos cercanos, lo que facilitara su filtrado en caso de cometerse estos fallos.

Por otra parte, los objetos del fondo son más sensibles a errores, ya que una pequeña variación en el cálculo de la disparidad puede resultar en mayores errores en metros. No obstante, los peatones que están tan lejos no son tan relevantes como los que están directamente delante de los vehículos.

También tenemos que tener en cuenta que durante el filtrado se eliminan valores que resultan ser demasiado elevados, por lo que estos errores además de ser menos comunes son menos relevantes y más fáciles de identificar.

Estos datos, aunque no son extremadamente precisos, ya que hay métodos que logran bajar del 2% de error según estos benchmarks, están dentro de unos límites razonables. Estamos obteniendo casi el 90% del mapa con errores menores al 5%, lo que supone un buen resultado para el hardware utilizado. [23]

En el apartado de tiempos, durante la ejecución el programa con el hardware de Nvidia se han obtenido tiempos bastante altos, del orden de 2 segundos. Concretamente durante la ejecución de la serie completa se ha obtenido una media de 2196 ms. Estos son tiempos referidos únicamente al cálculo del mapa y posterior conversión a metros de las medidas, pero suponen una gran parte del tiempo de ejecución, concretamente el 87% de media de un ciclo completo del programa.

En el benchmarck también se comprobaron tiempos con el procesador que se ha indicado, obteniendo un tiempo estimado de 1.1 segundos, casi la mitad del obtenido en nuestro proceso. Esto puede ser debido a que, aunque el hardware está diseñado para proyectos de visión artificial, no está cerca de ser el sistema más potente que se pueda tener ahora mismo. El sistema tiene 256 *CUDA Cores* que son los que se encargan de realizar los cálculos y trabajan a una frecuencia de 2GHz, lo que no se puede comparar directamente con las especificaciones de los test por ser distinto tipo de procesadores. Sin embargo, podemos ver en la página de Nvidia como esta misma compañía dispone de tarjetas gráficas dedicadas para este tipo de cálculos que superan en más de un orden de magnitud al número de núcleos de procesamiento a unas velocidades de funcionamiento similares o superiores a las del sistema empleado.

Esto nos hacer ver que, aunque el método de cálculo de distancias no es el punto fuerte del proyecto por su velocidad, su precisión está a la altura de la precisión esperada. Además, si sería posible su implementación si se dispusiera de un hardware lo suficientemente potente como para que se procesara en décimas de segundo en lugar de segundos completos, sin la necesidad de cambiar de método para calcularlas. [13] [24]

5.4 Sistema filtrado

El análisis de este último apartado se analizará el sistema de filtrado empleado para refinar los datos que se han obtenido durante las otras etapas. Este proceso consistirá en determinar si la clasificación obtenida de todos los puntos obtenida mediante el filtrado se corresponde con lo que realmente es en la imagen.

Para realizar este proceso se implementado en el código una nueva capa de superposición para poder determinarlo de forma visual, en la que solo se mostrarán los puntos correspondientes a la imagen actual, y que tendrán dos colores dependiendo de su clasificación. Este sistema ya se ha podido observar en las *figuras 17 y 18* que aparecen en la sección donde se explica el funcionamiento del filtrado.

Para realizar el análisis de estos resultados se ha tenido que hacer de forma manual revisando las 180 imágenes punto a punto. Esto es así, porque no se ha podido emplear ninguna técnica que nos permita determinar qué zonas pertenecen a los peatones y cuales no de forma precisa pixel a pixel como podría haber sido la segmentación semántica de imágenes.

Para esta revisión se han tenido que tomar ciertas consideraciones a la hora de clasificar los puntos en determinadas circunstancias.

- En primer lugar, los bordes de los peatones son zonas conflictivas. Esto se debe a que no se puede saber solo con mirar la imagen si la medida de distancia que estaremos obteniendo es del propio peatón o del fondo sobre el que está situado. Se han comprobado distintos bordes y no es consistente, el sistema de cálculo de distancias puede coger el valor del peatón o del fondo dependiendo de la colocación precisa en pixeles del punto. Por este motivo y sabiendo que el sistema de filtrado cumple su función de eliminar puntos que se desvíen bastante de la media, los puntos situados en fronteras de peatón con fondo se han considerado como puntos filtrados adecuadamente, tanto si se eliminan, como si pasan como válidos.

- En la secuencia de imágenes hay algunas que no se van a contabilizar los resultados hacia el número de efectividad final. Esto es así ya que resulta muy difícil cuantificar los niveles de acierto al clasificar los puntos. Estos casos son en su mayoría cuando un objeto se interpone entre el peatón detectado y la cámara. El problema es que muchos puntos se pierden y pasan a lugares de la imagen que no les corresponden, y los que se quedan en el objeto interpuesto llegan a ser tantos que la media de distancia le identifica como el punto de interés y elimina todos los demás que corresponden a los peatones. Esto solo ocurre hasta la próxima reinicializarían, por lo que como mucho serán grupos de 2 imágenes.

- Otro motivo por el que se han descartado algunas imágenes es bastante similar al anterior. Se trata de identificaciones de peatones que se encuentran parcialmente ocultos tras un vehículo aparcado, o situaciones en las que, aunque se está detectando el peatón, la Boundingbox que le corresponde tiene mucha más superficie sobre el fondo o al objeto que se interpone que al propio peatón. Cuando esto ocurre la cantidad de puntos que se sitúan sobre el peatón es en general menor que la que no está sobre él, por lo que al calcular la media para filtrar los datos, se consideran como datos erróneos aquellos que contienen en al peatón invirtiéndose los resultados esperados.

  Hay que mencionar que cuando esto ocurre solo se descarta si ocurre a todos los peatones de la imagen. Si en su lugar solo ocurre a una pequeña porción de ellos, simplemente no se contabilizan esos puntos, para tener la mayor cantidad de datos posibles para hacer el análisis.

- El ultimo motivo por el que se han descartado algunas imágenes es porque los puntos no se han reubicado en las regiones adecuadas al calcular el flujo óptico. Esto ocurre principalmente cuando los peatones y sus puntos asociados están saliendo de la escena. El sistema de flujo no es capaz de situarlos porque ya están fuera de la imagen y se terminan esparciendo por la imagen de modo que se arruinan las medidas con el filtrado de ese determinado peatón.

  Este fallo ocurre también en un punto del recorrido con un ciclista que circula por la calzada en dirección contraria. Debido a que la velocidad relativa al coche es mucho más alta que la de los peatones, independientemente de su sentido, y que se encuentra a una distancia muy corta, los puntos se pierden detrás de él ya que se desplazan grandes distancias en la imagen y el sistema de flujo óptico no está diseñado para movimientos tan grandes en una sola imagen.

- Como última nota antes de pasar a analizar los puntos, tampoco se han contabilizado las detecciones erróneas de peatones que se han mencionado en el apartado de detección. Aun así, las imágenes que las contenían solo se han descartado cuando tenía otros de los errores importantes de los que se acaban de mencionar, en otros casos solo se han dejado a un a lado para no contabilizar los puntos concretos que constituyen la detección errónea.

Para realizar el análisis de los datos que se han obtenido, se han clasificado en dos tipos, positivos y negativos. Los positivos son aquellos que son datos válidos y los negativos todos aquellos que representan datos que no pertenecen a peatones. A su vez cada uno de ellos pueden ser verdaderos o falsos, dependiendo de si se han clasificado adecuadamente o no.

Con la salida del programa solo obtenemos los datos clasificados en positivos y negativos, representados en verde y rojo para una mejor visualización. El proceso que se ha hecho de forma manual es determinar si realmente están bien clasificados o si por el contrario representan un falso caso.

Así nos quedara un espacio muestral que se puede ver representado en la siguiente figura. Cada mitad del recuadro representa los datos clasificados como positivos y negativos a izquierda y derecha respectivamente. Sin embargo, los datos contenidos en el círculo central representan aquellos datos que se han clasificado de un modo pero que están mal clasificados y corresponderían al otro grupo, lo que se corresponde con la tarea de reidentificación manual.

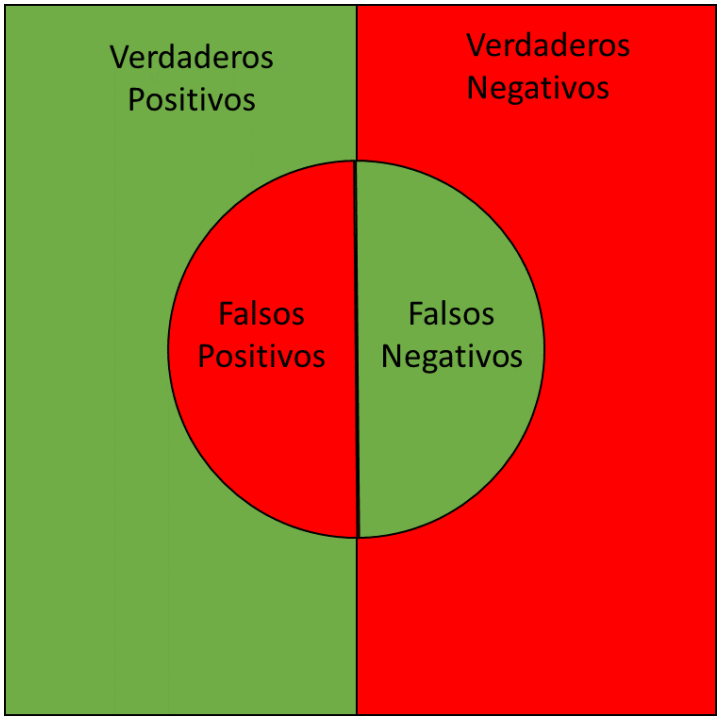

Fig. 5.6 Categorías de clasificación de los datos obtenidos

El sistema que se utilizara para cuantificar el acierto del sistema se denomina ***precision and recall***, en español ***precisión y exhaustividad***. Este sistema es el que se emplea para determinar la calidad de los resultados de procesos de búsqueda y extracción de la información, muy utilizado en los sistemas de aprendizaje automático. [25]

Para calcularlo se debe hacer con las fórmulas que se describen a continuación.

$$Precision = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ positivos}$$

$$Exhaustividad = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Negativos}$$

Con la fórmula de la precisión, la pregunta a la que tratamos dar respuesta es a cuál es la proporción de identificaciones positivas es correcta. La fórmula de exhaustividad por su parte, nos responde a otra realidad, la proporción de Verdaderos Positivos identificados correctamente.

En las próximas páginas se presentarán y analizarán todos los datos que se han podido extraer de la secuencia de imágenes tras ser procesadas por el programa y su posterior revisión manual, además de otras conclusiones más generales obtenidas de los mismos datos.

En primer lugar, tenemos recogido el análisis del número de imágenes que finalmente se han empleado tras descartar las que no cumplían los requerimientos que se han estipulado.

	Validas	**Descartadas**	**Sin Datos**	**Total**
Imágenes	124	44	12	180
Porcentajes (%)	68,89	24,44	6,67	100,00

Tabla 4 Resumen de Imágenes del set

Al final después de analizar las imágenes en busca de los defectos que se han mencionado, se han terminado descartando 43 imágenes del total de 180. A estas hay que sumar 12 en las cuales no se ha podido aplicar ningún tratamiento, debido a que no se tenían puntos en las imágenes. Estas 12 imágenes se corresponden a cuatro imágenes en las que el sistema de identificación de peatones no detecto ninguno y como consecuencia las dos imágenes siguientes hasta que se volvió a ejecutar el algoritmo de detección.

Con esto nos quedan un total de 124 imágenes en las que sí que se han podido realizar los análisis del filtrado de puntos. Aunque no es un dato extremadamente alto en porcentaje, casi un 70%, nos deja una muestra lo suficientemente grande como para poder evaluar los resultados que se han extraído y sacar conclusiones del proyecto.

Por tanto, como siguiente paso, se analizará la clasificación realizada por el sistema durante el proceso de filtrado.

De la salida del programa podemos obtener el total de positivos y negativos detectados. Los falsos tanto positivos como negativos son el resultado de contar de forma manual cuantos de cada tipo se han identificado mal. Los verdaderos simplemente los podemos obtener con una simple resta del número total identificado por el sistema.

En la tabla inferior, se tienen todos los puntos de las 124 imágenes que se han considerado. Suman un total de casi 9500 puntos entre positivos y negativos.

	Verdaderos	Falsos	Total
Positivos	6075	338	6413
Negativos	2905	139	3044
Total	8980	477	9457

Tabla 5 Resumen de los puntos totales obtenidos tras filtrado

Se puede apreciar como el número de falsos casos, tanto de positivos como de negativos, es significativamente inferior al de verdaderos casos. Esto se puede apreciar mejor en el gráfico a continuación.

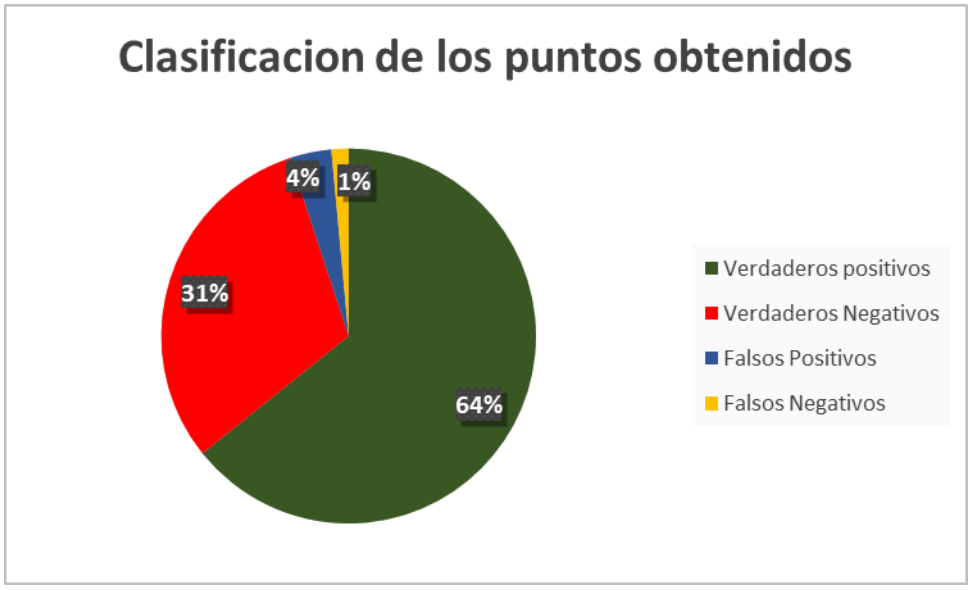

Fig. 5.7 Gráfico con la clasificación de los puntos obtenidos tras el filtrado

Podemos ver como los Verdaderos, los puntos que se han identificado correctamente suponen el 95% de todos los puntos identificados. Así tenemos que tan solo el 5% del total es identificado de forma errónea.

De los datos mal clasificados, tan solo el 1% del total es el que representa los falsos negativos, esto significa que solo 1 de cada 100 puntos que se tienen son datos que pertenecen a un peatón y son eliminados por error. Este es un dato muy bueno, ya que supone que se tiene un caso en el que prácticamente no se está perdiendo nada de información potencialmente útil para el sistema.

El dato de 4% de falsos positivos, aunque ligeramente superior al anterior, también es muy bueno. Nos indica que no estamos dejando pasar mucha información no deseada hasta las salidas del sistema.

Este caso es mejor que si los valores estuviesen invertidos, ya que de este modo se podrá seguir mejorando con otras técnicas para seguir refinado los datos sin haber perdido antes mucha información útil.

Estos datos los podemos contextualizar analizando la media de los errores cometidos por imagen y tipo de clasificación. Esto se muestra en la tabla 6 calculada imagen por imagen, y luego haciendo la media de los porcentajes obtenidos y también de todos los datos de una sola vez.

	Positivos	**Negativos**
Error medio por imagen (%)	5,29	4,30
Error medio total (%)	5,27	4,57

Tabla 6 Resumen errores cometidos al clasificar puntos de interés

En ambos casos tenemos que los valores medios de error cometido al clasificar los puntos rondan el 5% lo que se puede considerar como un muy buen resultado. Además, no existen diferencias sustanciales entre el cálculo por imagen y el total, lo que nos indica que no existe una gran desigualdad en ciertas imágenes. Esto se puede corroborar al comprobare el número de imágenes con una tasa de error mayor del 15% es de tan solo 15 en el caso de los falsos positivos y 18 en el de los falsos negativos.

Ahora se pasará a calcular los parámetros de precisión y exhaustividad que se han mencionado anteriormente. Para poder seguir comprobando la consistencia de los datos, se ha vuelto a calcular de las dos maneras, de forma individual a cada imagen y luego calculan su media, y también calculando los parámetros con todos los datos disponibles a la vez como una única muestra. Estos datos junto con la diferencia entre ambos se muestran en la siguiente tabla.

	Precisión	**Exhaustividad**
Datos Totales (%)	94,73	97,76
Media imágenes analizadas (%)	94,51	97,21
Diferencia (%)	0,22	0,56

Tabla 7 Resultados de precisión y exhaustividad de las imágenes analizadas

Podemos ver como los datos que se obtienen son muy altos, lo que significa que la calidad de los resultados es muy alta de acuerdo a este criterio. Además, este valor de alrededor de 94,5 % para

la precisan y 97 % para la exhaustividad es consistente en ambas medidas. Esto refuerza la idea de que no hay muchas imágenes que sobresalgan en exceso de la media.

Estos datos son los que se han obtenido tras descartar ciertas imágenes en las que otros sistemas del propio programa no han sido capaces de dar buenos resultados, de esta forma hemos podido comprobar que la salida del programa en condiciones ideales es muy eficiente. Sin embargo, podemos calcular los valores de precisión y exhaustividad de la secuencia completa, sin descartes. Si a las imágenes a las que no se han podido determinar los casos de positivos y negativos, se les asigna un valor en ambos criterios de cero, podemos ver los siguientes resultados.

	Precisión	**Exhaustividad**
Media todas las imágenes (%)	70,17	72,18
Media imágenes analizadas (%)	94,51	97,21
Diferencia (%)	24,33	25,03

Tabla 8 Comparativa de precisión y exhaustividad entre subconjuntos de imágenes

Se aprecia como los valores son significativamente más bajos habiendo disminuido un 25% en ambos criterios. No obstante, seguimos teniendo valores de precisión y exhaustividad del entorno del 70 %. Esto es algo remarcable, ya que estamos puntuando a las imágenes que se han desechado con la peor puntuación posible en los índices y aun así el resultado sigue siendo un valor bastante alto.

Finalmente, aunque este valor no tenga mucho sentido práctico nos da un valor que clasifique el trabajo del todo el conjunto, ya que podemos decir que en un 70% de los casos el programa hace una muy buena labor extrayendo la información de la que se dispone incluso en condiciones muy difíciles para los sistemas de reconocimiento óptico que se han empleado.

5.5 Tiempos de ejecución

En este último apartado se va a analizar un parámetro muy importante para la ejecución de programas enfocados a un empleo en tiempo real, el tiempo de ejecución.

Para medir estos tiempos de ejecución se ha hecho uso de las librerías *time.h* que nos permiten obtener los valores de los ciclos de procesador que han sido necesarios para la ejecución que se han utilizado. Esta información por sí sola no nos aportara nada, pero sabiendo la velocidad de procesamiento podemos transformarlo a milisegundos que podremos utilizar para cuantificar los resultados.

Antes de ir a los resultados hay que remarcar que son los tiempos que se obtienen con el hardware Jetson-TX2. Por lo general cualquier hardware más potente con mayores capacidades de cálculo obtendrían tiempos más bajos siempre que se trate de una GPU especializada en este tipo de trabajos.

Para el análisis de tiempos tenemos que tener en cuenta que el programa tiene un primer proceso de inicialización donde se cargan todos los datos necesarios para el proceso de detección de peatones. Tras este proceso, se da comienzo a la ejecución cíclica en la que se extraen y procesan los datos de las imágenes. Este proceso cíclico se ha subdividido a su vez en las cuatro fases que se han descrito en capítulos anteriores para poder analízalas de forma independiente, además de una fase adicional para el guardado de los datos en disco.

Para la fase de carga de datos se ha obtenido un tiempo medio entre distintas ejecuciones del programa de 2400 ms. Este es un tiempo que, aunque es necesario para poder ejecutar el núcleo del programa, no se va a considerar en ningún otro aspecto, ya que se contabiliza como tiempo de puesta a punto y no es relevante para la ejecución en tiempo real, solo supone un tiempo mínimo que hay que esperar para que el sistema este operativo.

El resto de tiempos de la parte repetitiva del programa se muestran en la siguiente tabla. Los datos están anotados en milisegundos, y son la media de los ciclos de inicialización donde se buscan nuevos peatones, y los que mediante el flujo óptico solo se limitan a reubicar los puntos existentes en las nuevas imágenes. Además, se incluye una tercera fila para poder tener una mayor claridad al representar las diferencias entre ambas medidas.

	Detección de peatones	**Asignación puntos críticos**	**Calculo de distancias**	**Filtrado de datos**	**Guardado en disco**	**Total**
Media ciclo de inicialización	180,28	55,22	2196,72	75,23	150,10	2657,55
Media ciclo de no inicialización	19,19	14,41	2195,68	72,58	252,93	2448,61
Diferencia	161,09	40,81	1,04	2,65	3,35	208,94

Tabla 9 Resumen de tiempos de la aplicación

De la tabla se pueden extraer dos conclusiones principales.

- La primera es que existe una diferencia notable entre los tiempos de los ciclos de inicialización y los que no lo son. Esto era esperable, ya que el proceso de búsqueda de peatones solo se realiza durante la inicialización, este era uno de los motivos por los que no se inicializaba en todos los ciclos. Esta diferencia asciende a 208,94 milisegundos de media, lo que no es un tiempo despreciable si queremos ejecutar el algoritmo varias veces por segundo.

  Esta diferencia de tiempo se encuentra distribuida únicamente entre los dos primeros procesos que se ejecutan en cada ciclo, ya que no existe diferencias notables entre ambos tipos de ciclos, por lo que las diferencias de tiempo se pueden considerar como desviaciones estadísticas. Este no es el caso sin embargo de los apartados de detección de peatones, y de asignación de puntos críticos a la imagen.

En el primero de ellos, el módulo de detección de patones, la diferencia asciende a algo más de 160 milisegundos, lo que supone casi 8 veces el tiempo de los ciclos que no se inicializa. Esto se debe a que cuando no se necesita inicializar, el sistema de detección de peatones de Jetson inference no interviene, y el tiempo que se está reflejando es solamente el tiempo de carga de las imágenes e inicialización de variables que se emplean en el resto del programa.

sin embargo, en el apartado de asignación de puntos la diferencia no es tan grande, siendo solo de 40 milisegundos. Esta diferencia era esperable, y se debe a que cuando se inicializan los peatones, se ejecuta el algoritmo de reasignación de puntos, *goodFeaturesToTrack*, además del de seguimiento de los existentes, por lo que la carga de cálculo es ligeramente superior.

- El segundo aspecto importante que destaca en los datos es la duración media del cálculo de distancias. Este tiempo engloba solo la creación del mapa de disparidad y la conversión de esos datos a metros para cada punto en cada imagen, sin embargo, emplea más del 80% del tiempo medio de ciclo. Esto no sería un gran problema si no fuera porque este tiempo es aproximadamente 2,1 segundos. Esto significa que la frecuencia a la que se podría ejecutar el programa es como mucho 0,4 Hz.

  Si tenemos en cuenta otros estudios sobre vehículos autónomos, vemos que las frecuencias a las que suelen actualizar los datos son de entre 5 y 10 veces por segundo, por lo que nuestros tiempos están un orden de magnitud por encima de estos estándares. [26] [27]

Se puede decir por tanto, que es el sistema de cálculo de distancias el que nos está ralentizando mucho la ejecución del programa. Por este motivo, vamos a analizar si es la única función que está lastrando el programa, o si por el contrario es el único punto que habría que intentar acelerar. Para ellos se ha representado la siguiente gráfica, y la tabla que contiene sus valores. En estas se representan todos los tiempos del sistema a excepción del cálculo de distancias.

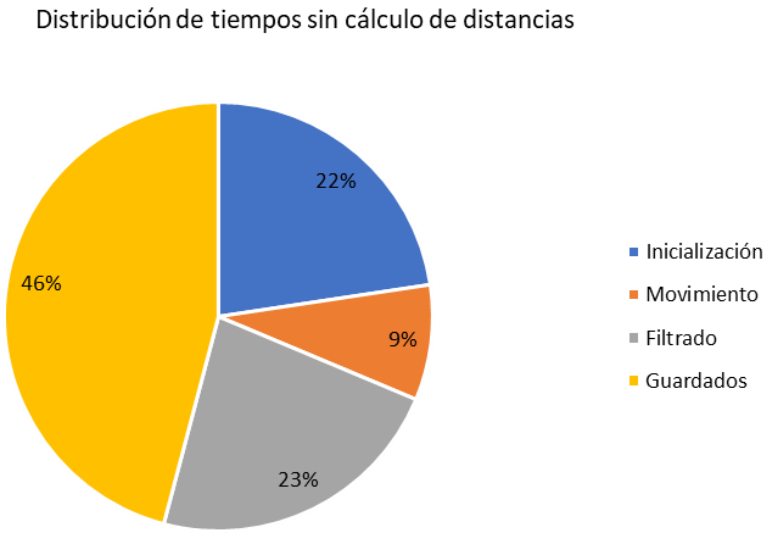

Fig. 5.8 Gráfico con la distribución de tiempos sin cálculo de distancias

	Detección de peatones	Asignación puntos críticos	Filtrado de datos	Guardado en disco	Total
Media ciclo (ms)	72,89	28,01	73,47	147,87	322,23
Porcentaje	22,62	8,69	22,80	45,89	100,00

Tabla 10 Resumen de tiempos de la aplicación sin cálculo de distancias

Se puede observar cómo, aunque los tiempos sin el cálculo de distancias solo permitirían unos 3 ciclos por segundo, estos son mucho menores que los anteriores.

Ahora el único aspecto que destaca sobre los demás es el guardado en disco de la información. No obstante, este tiempo se puede reducir hasta el punto de eliminarlo si no necesitamos guardar todas las imágenes para poder analizarlas a posteriori. Este guardado no sería necesario durante una aplicación real del software. Los únicos datos relevantes son las distancias y desplazamientos de los peatones, y eso se puede pasar directamente al programa que los vaya a aprovechar mediante variables, o simplemente escribirlos en un documento de texto si es un programa externo, lo que prácticamente no requiere de recursos en comparación con guardar una imagen completa.

De esta forma, se pueden repetir el análisis de tiempos de la secuencia de imágenes suprimiendo todas las líneas de código que sean salidas que no son estrictamente necesarias para obtener el resultado buscado.

Tras esto se pudo observar como se obtienen significativos recortes en los tiempos de ciclo, especialmente en el filtrado de datos. Esto se debe a que ya no tenemos que representar los puntos a cada pasada sobre la imagen y tras la última de ellas guardar la imagen a disco con la representación de los puntos.

Además, se ha suprimido toda la información que se mostraba por consola ya que solo es útil durante la depuración de problemas de ejecución, con la finalidad de saber que procesos están siendo ejecutados correctamente y cuáles no.

Los datos de tiempos medios de las 180 imágenes suprimiendo las salidas de verificación del sistema y sin tener en cuenta el proceso de cálculo de distancias se recogen en la siguiente tabla.

	Detección de peatones	Asignación puntos críticos	Filtrado de datos	Total
Media ciclo (ms)	62,35	26,52	18,37	107,23
Porcentaje	58,14	24,73	17,13	100,00

Tabla 11 Resumen de tiempos de la aplicación sin cálculo de distancias y sin visualización de resultados

Se puede ver ahora que los tiempos han descendido hasta los 107 milisegundos por ciclo, lo que supone una reducción de más de un tercio del tiempo respecto a los anteriores con las salidas de imágenes.

Esto implica que si el programa solo realizara estos procesos se podría ejecutar de media a unos 9 ciclos por segundo, lo que lo dejaría en la zona alta de las recomendaciones para estos sistemas autónomos.

Sin embargo, no podemos olvidar que estos datos no tienen en cuenta el proceso de cálculo de distancias, por lo que si suponemos que queremos alcanzar los 5 Hz nos dejaría con aproximadamente 93 milisegundos por ciclo para ejecutar el cálculo de esas distancias.

Este tiempo esta algo más de un orden de magnitud por debajo de los tiempos obtenidos para el módulo de cálculo distancias en el programa. Esta parte del programa es el cuello de botella de la aplicación, es el elemento que nos está impidiendo mejorar los rendimientos de tiempo y por tanto el punto donde se deberían centrar los esfuerzos en futuros trabajos para la mejora del sistema, ya sea adquiriendo un sistema más rápido, optimizando el algoritmo empleado o un conjunto de ambas.

CAPÍTULO 6: Presupuesto

En este sexto capítulo se va a presentar una estimación de los costes en los que se incurriría para el desarrollo de este proyecto. En él se van a incluir tanto los costes de los materiales y productos que han sido necesarios, así como el trabajo realizado para obtener los resultados.

En primer lugar analizaremos el hardware empleado. En este apartado tenemos que considerar que la plataforma principal de todo el proyecto es la Jetson Tx2, donde se ha desarrollado todo el código del programa. Este es un pequeño ordenador, que cuenta con salidas y entradas, pero que no cuenta con los periféricos necesarios para ellos por lo que también habrá que incluirlos, a diferencia de si hubiésemos empleado un sistema completo como podría ser un portátil.

Este sistema aunque no está diseñado como estación de trabajo, se pude llegar a usar para ello en labores sencillas como la edición de texto o la búsqueda de información en línea. Por este motivo, no sería estrictamente necesario el empleo de otro ordenador auxiliar para estas tareas de las que también se ha requerido durante el proyecto. Además, al utilizar software libre los programas que se han requerido no han supuesto un coste adicional.

En la siguiente tabla se recogen todos los gastos referidos al equipamiento que se ha empleado.

Elemento	Descripción	Coste del producto	cantidad	Coste final en Euros (€)
Jetson TX2	Sistema de principal de procesamiento del software, pequeño ordenador.	609.46 $	1	526,69
Periféricos de entrada	Teclado y ratón necesarios para poder hacer uso del sistema	50 €	1	50
Pantalla	Dispositivo de visualización compatible con el sistema empleado	80	1	80
Total hardware				**659,69**

Tabla 12 Costes de hardware del proyecto

Hay que destacar que dado que el coste de la Jetson TX2 es una conversión de divisas desde el dólar al euro debido a que la compañía Nvidia es de Estados Unidos. Además, los costes de los periféricos son una estimación basada en material de oficina de la tienda online Amazon.

En segundo lugar, tenemos que realizar las estimaciones del coste de personal necesario para el desarrollo completo de la aplicación. Este coste se realizará haciendo un recuento de las horas empleadas para la realización de cada una de las tareas del proyecto, calculando posteriormente el precio. Para ello se tomará como salario para un ingeniero industrial un salario de 18 €/hora.

Descripción de la actividad	Horas empleadas (h)	Coste (€)
Curso de iniciación a la visión artificial	50	900
Planificación y diseño previo	20	360
Instalación de librerías y familiarización con Ubuntu	30	540
Desarrollo del código	350	6300
Redacción de la memoria	130	2340
Total	**580**	**10.440**

Tabla 13 Costes y tiempos de las horas de trabajo del proyecto

Una visión más detallada de las horas dedicadas a cada una de estas fases, se puede ver en la siguiente tabla. En ella se detallan las horas que finalmente fueron necesarias para realizar las tareas principales que se habían planificado originalmente en el *apartado 1.4*, planificación del proyecto, presente en el primer capítulo de esta memoria.

Periodo diciembre 2017 - agosto 2018	Diciembre	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	**Total**
Curso de iniciación a la visión artificial	20	30								**50**
Instalación de librerías y familiarización con Ubuntu	10	10								**30**
Planificación y diseño previo			30							**20**
Desarrollo del código				40	40	30	120	120		**350**
Redacción de la memoria							20	30	80	**130**
Total mensual	**30**	**40**	**30**	**40**	**40**	**30**	**140**	**150**	**80**	**580**

Tabla 14 Resumen de horas empleadas por tareas y meses

Aunque estas no se ajustan perfectamente e a lo inicialmente establecido en la planificación, si que respetan las fechas críticas que se habían marcado, lo que ha supuesto que se finalizara a tiempo.

Así, finalmente se tiene una estimación final del coste total del proyecto, teniendo en cuenta tanto el material empleado como las horas de trabajo necesarias para llevarlo a cabo.

El resultado final del proyecto asciende a 11099,69 y se encuentra reflejado en esta última tabla.

Descripción de la actividad	Coste (€)
Material necesario para el proyecto, Hardware utilizado	659,69
Mano de obra necesaria para realizar el proyecto y la posterior memoria	10.440
Total	**11099,69**

Tabla 15 Coste final del proyecto

CAPÍTULO 7: Conclusiones

En este último capítulo de la memoria se va a resumir el programa en su conjunto viendo qué objetivos se han cumplido y cuáles no. Además, se propondrán posibles mejoras y soluciones a los problemas que no se han conseguido solventar o aquellos aspectos en los que se podrían haber obtenido unos resultados mejorables.

7.1 Objetivos cumplidos

Como resultado final, el programa que se ha desarrollado es un software que cumple con su cometido de identificación de peatones en entornos de conducción autónoma por vías urbanas.

El programa logra todos los objetivos que se proponían en un inicio y se puede usar de forma independiente. Este consigue proporcionar datos de posición y desplazamiento de los peatones que monitoriza.

Sin embargo, este programa no pretende ser un producto definitivo y estático para un uso independiente. Todo lo contrario, su objetivo último seria formar parte de un conjunto mayor en un sistema totalmente autónomo que lo integraría como uno de sus sistemas de seguridad y control.

Los problemas a los que finalmente hace frente y consigue solucionar son los que se enumeran a continuación.

- En primer lugar, el proyecto se había impuesto la limitación de hacer uso única y exclusivamente de un tipo de sensor. De esta forma se aumentaría la versatilidad del sistema manteniéndolo accesible para su empleo en sistemas físicos. Para hacer esto se optó por cámaras convencionales ya que serían capaces de obtener los datos que se requerirían al proyecto gracias a su polivalencia. Mediante el uso de una pareja de estas cámaras en paralelo, se puede extraer todos los datos necesarios. Además de ser a su vez un sistema relativamente accesible y muy extendido en el mundo de la conducción autónoma.
- El segundo punto que se ha logrado alcanzar es poder utilizar la información que se extrae de los sensores empleados para obtener la información buscada. No basta con ser capaces de obtener toda la información posible, también que hay que ser capaces de filtrarla e interpretarla. En este caso lo que buscábamos era poder identificar peatones en el entorno de un vehículo entre todos los demás cuerpos que se podían presentar, por lo que también lo podemos considerar un éxito.

- El tercero, que es el paso natural al anterior, es que es capaz de extraer información concreta de cada uno de los peatones que se han identificado. En particular la posición y los desplazamientos que están asociados a cada uno de ellos. Estos datos han sido tratados y filtrados para complementar la información obtenida directamente de las cámaras para hacerla lo más precisa posible. Estos se presentan a la salida de forma ordenada y clara para facilitar su uso en otras aplicaciones.

Al final tenemos un programa que es capaz de realizar la detección y seguimiento de los peatones presentes en el entorno del vehículo mediante el análisis de los datos ópticos de las cámaras, determinando sus parámetros de posición y desplazamiento relativo al vehículo todo ello en el entorno urbano.

7.2 Líneas de mejora

Aunque como ya se ha comentado se han cumplido los objetivos plateados para este proyecto, el sistema tiene varias líneas de mejora en las que se podría progresar y mejorar. Esto permitiría tener mejores tasas de acierto en general, obteniendo resultados más precisos en situaciones difíciles para los sistemas automáticos. Además, se podría progresar bajando los tiempos de ejecución para tener mayores tasas de refresco para los datos tanto de entrada como de salida.

Estos aspectos se pueden resumir en dos puntos principales.

- El primero de estos puntos es el sistema de reconocimiento de peatones, especialmente en las circunstancias adversas de iluminación que se han descrito anteriormente. Aunque en general el sistema ha tenido una muy buena respuesta al complejo entorno que se le ha presentado, han destacado situaciones en las que los resultados eran mejorables.

  Estas condiciones de iluminación tan complicadas que estaban presentes en gran parte de las imágenes de la muestra hacían que peatones que para el ojo humano fueran relativamente fáciles de identificar, el sistema empleado no los reconociera y por tanto pasaran inadvertido para el programa.

  Para poder mejorar este aspecto se debería crear otra red de identificación o mejorar la existente. Esta debería estar enfocada a situaciones más complejas como la mencionada de iluminación dispar en zonas de una misma imagen. También se podría poner un énfasis adicional en intentar entrenar al sistema en reconocer peatones que estén parcialmente cubiertos por otros objetos, lo que haría que el sistema fuera muy completo y se pudiera acercar más al nivel de identificación del que nosotros como personas somos capaces de alcanzar.

- El otro aspecto en el que también se podría mejorar bastante es en los tiempos de ejecución de los ciclos del programa. Mas concretamente en la parte relativa al cálculo de distancias. Para lograr esto se podrían seguir varios caminos.

  El primero y más radical seria cambiar el módulo por completo por otro algoritmo más rápido y que siguiera manteniendo o mejorara las tasas de acierto. Esto aunque a priori es más complejo, podría llevarse a cabo si se quiere seguir mejorando este aspecto, creando desde cero dicho algoritmo.

  Otra aproximación más conservadora, sería hacer modificaciones al algoritmo empleado para no realizar un mapa de disparidad de toda la imagen, sino solo de las regiones que hemos considerado de interés en cada imagen. Esto haría que la región de trabajo del algoritmo fuera mucho más reducida, por lo que los requerimientos de cálculo se reducirían notablemente, y con ellos el tiempo empleado en obtenerlos.

  La última opción que se podría considerar seria realizar una actualización del hardware por otro más potente. Aunque esto dependería del presupuesto del que se dispusiese, un equipo más potente sería capaz de reducir tiempos con un mismo programa. Sin embargo, puede que esta última opción no fuera siempre posible debido a las limitaciones técnicas que se tengan.

  Sin embargo como última nota, aunque cualquiera de las tres vías seria valida, sin duda el método para obtener unos tiempos mucho más bajos seria realizarlas todas en mayor o menor medida. Aplicando todos ellos a la vez se lograría aprovechar las mejoras de cada uno de ellos y de este modo se alcanzaría una mayor reducción en los tiempos finales del proyecto.

CAPÍTULO 8: Bibliografía

[1] European Road Safety Observatory, "Traffic Safety Basic Facts 2017", Junio del 2017. [En línea]. Disponible en: https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/statistics/dacota/bfs2017_pedestrians.pdf [Consultado: 15-07-2018].

[2] Govtack, "H.R. 3388: SELF DRIVE Act". [En línea]. Disponible en: https://www.govtrack.us/congress/bills/115/hr3388/summary#oursummary [Consultado: 28-08-2018].

[3] "Communication from the commission to the European Parliament, the Council, the European Economic and Social Committee, the Committee of the Regions On the road to automated mobility: An EU strategy for mobility of the future". Comisión Europea. Bruselas, 17-05-2018. [En línea]. Disponible en: https://ec.europa.eu/transport/sites/transport/files/3rd-mobility-pack/com20180283_en.pdf [Consultado: 28-08-2018].

[4] Dirección general de tráfico, "Tráfico establece el marco para la realización de pruebas con vehículos de conducción automatizada en vías abiertas a la circulación" . [En línea]. Disponible en: http://www.dgt.es/es/prensa/notas-de-prensa/2015/20151116-trafico-establece-marco-realizacion-pruebas-vehiculos-conduccion-automatizada-vias-abiertas-circulacion.shtml [Consultado: 28-08-2018].

[5] Hipertextual , "¿Qué sensores tiene un coche autónomo y cómo funcionan?", *Hipertextual, 2-6-2017*. [En línea]. Disponible en: https://hipertextual.com/presentado-por/ford/sensores-coches-autonomos [Consultado: 20-07-2018].

[6] D. Kohanbash, "LIDAR vs RADAR: A Detailed Comparison", *Robotsforroboticists*, 04-05-2017. [En línea]. Disponible en: *http://robotsforroboticists.com/lidar-vs-radar/* [Consultado: 20-07-2018].

[7] National Highway Traffic Safety Administration, "Automated vehicles for safety". [En línea]. Disponible en: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety [Consultado: 15-07-2018].

[8] A. Pérez, "¿Qué es la conducción autónoma? Todo lo que necesitas saber", *periodismodelmotor*, 01-08-2017. [En línea]. Disponible en: https://periodismodelmotor.com/que-es-conduccion-autonoma-autopilot/169482/[Consultado: Consulta: 19-07-2018].

[9] European automobile manufacturers association, "Passenger Cars World". [En línea]. Disponible en: https://www.acea.be/statistics/tag/category/passenger-cars-world%20 [Consultado: 15-07-2018].

[10] Dirección general de tráfico, "Tablas estadísticas 2016", 2016. [En línea]. Disponible en: http://www.dgt.es/es/seguridad-vial/estadisticas-e-indicadores/parque-vehiculos/tablas-estadisticas/2016/ [Consultado: 16-07-2018].

[11] Open Source Computer Vision, "OpenCV modules", 23-12-2016. [En línea]. Disponible en: https://docs.opencv.org/3.2.0/ [Consultado: 15-02-2018].

[12] EDX, "Introducción a la visión por computador: desarrollo de aplicaciones con OpenCV". [Curso en línea]. Disponible en: https://www.edx.org/es/course/introduccion-la-vision-por-computador-uc3mx-isa-1x-0# [Consultado: 27-01-2018].

[13] Nvidia, "Nvidia Jetson". [En línea]. Disponible en: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/ [Consultado: 30-02-2018].

[14] D. Franklin, C. YATO, D. Nguyen, "jetson-inference", *Github*. [En línea]. Disponible en: https://github.com/dusty-nv/jetson-inference [Consultado: 30-02-2018].

[15] Aprenderpython, "Detección De Esquinas Harris". [En línea]. Disponible en: https://www.aprenderpython.net/deteccion-esquinas-harris/ [Consultado: 15-03-2018].

[16] Open Source Computer Vision, "Feature Detection", 22-08-2018. [En línea]. Disponible en: https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=goodfeaturestotrack [Consultado: 30-08-2018].

[17] J. Bouguet, "Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker. Description of the algorithm", Intel Corporation, Microprocessor Research Labs. [En línea]. Disponible en:
https://pdfs.semanticscholar.org/aa97/2b40c0f8e20b07e02d1fd320bc7ebadfdfc7.pdf
[Consultado: 06-03-2018].

[18] Open Source Computer Vision," Motion Analysis and Object Tracking", 22-08-2018. [En línea]. Disponible en:
https://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=calcopticalflowpyrlk [Consultado: 05-03-2018].

[19] R. Washbourne, "What Is Stereo Matching?", DevPy, 2015. [En línea]. Disponible en: https://www.devpy.me/what-is-stereo-matching/ [Consultado: 28-03-2018].

[20] A. Geiger, P. Lenz, C. Stiller, R. Urtasun, "Vision meets Robotics: The KITTI Dataset", International Journal of Robotics Research, 2012. [En línea]. Disponible en: http://www.cvlibs.net/datasets/kitti/eval_scene_flow_detail.php?benchmark=stereo&result=67877bd6fcc43163421fa0108c7df83bbc69fea3 [Consultado: 02-04-2018].

[21] Open Source Computer Vision, "cv::StereoSGBM Class Reference", 25-8-2018. [En línea]. Disponible en: https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html [Consultado: 02-04-2018].

[22] Open Source Computer Vision, "Depth Map from Stereo Images", 18-5-2015. [En línea]. Disponible en: https://docs.opencv.org/3.1.0/dd/d53/tutorial_py_depthmap.html [Consultado: 02-04-2018].

[23] M. Moritz, G. Andreas, "Object Scene Flow for Autonomous Vehicles", Conference on Computer Vision and Pattern Recognition (CVPR), 2015. [En línea]. Disponible en: http://www.cvlibs.net/publications/Menze2015CVPR.pdf [Consultado: 15-04-2018].

[24] Nvidia, "Compare 10 series GPUs". [En línea]. Disponible en: https://www.nvidia.com/en-us/geforce/products/10series/compare/ [Consultado: 20-07-2018].

[25] Google developers, "Curso intensivo de aprendizaje automático". [En línea]. Disponible en: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall?hl=es-419 [Consultado: 20-07-2018].

[26] J. Pérez, Et al. "Comunicación entre vehículos autónomos en tiempo real, para maniobras de alto riesgo", en XXIX Jornadas de Automática, Tarragona, Septiembre 2008. [En línea]. Disponible en: http://intranet.ceautomatica.es/old/actividades/jornadas/XXIX/pdf/303.pdf [Consultado: 25-07-2018].

[27] S. Sukkarieh, E.M. Nebot, H. F. Durrant-Whyte, "A High Integrity IMU/GPS Navigation Loop for Autonomous Land Vehicle Applications", *IEEE transactions on robotics and automation, VOL.12, NUM. 3*, Junio 1999. [En línea]. Disponible en: http://www-personal.acfr.usyd.edu.au/nebot/publications/IEEE_ROB_gps_ins.pdf [Consultado:25-07-2018].